

This is a repository copy of *Compositional Assume-Guarantee Reasoning of Control Law Diagrams using UTP*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/129640/>

Monograph:

Ye, Kangfeng, Foster, Simon David orcid.org/0000-0002-9889-9514 and Woodcock, JAMES Charles Paul orcid.org/0000-0001-7955-2702 *Compositional Assume-Guarantee Reasoning of Control Law Diagrams using UTP*. Working Paper. (Unpublished)

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Compositional Assume-Guarantee Reasoning of Control Law Diagrams using UTP

Kangfeng Ye Simon Foster
Jim Woodcock
University of York, UK

{kangfeng.ye, simon.foster, jim.woodcock}@york.ac.uk

April 23, 2018

Abstract

This report is a summary of our work for the VeTSS funded project “Mechanised Assume-Guarantee Reasoning for Control Law Diagrams via Circus”. Our Assume-Guarantee (AG) reasoning of control law diagrams is based on Hoare and He’s Unifying Theories of Programming and their theory of designs. In this report, we present developed theories and laws to map discrete-time Simulink block diagrams to designs in UTP, calculate assumptions and guarantees, and verify properties for modelled systems. A practical application of our AG reasoning to an aircraft cabin pressure control subsystem is also presented. In addition, all mechanised theories in Isabelle/UTP are attached in Appendices. In the end of this report, we summarise current progress for each work package.

Contents

1	Introduction	4
2	Preliminaries	6
2.1	Control Law Diagrams and Simulink	6
2.2	Unifying Theories of Programming	7
2.2.1	Designs	7
3	Assumptions and General Procedure of Reasoning	9
3.1	Assumptions	9
3.2	General Procedure of Applying Assumption-Guarantee Reasoning	9
4	Semantic Translation of Blocks	10
4.1	State Space	10
4.2	Healthiness Condition: <i>SimBlock</i>	10
4.3	Blocks	11
4.3.1	Pattern	11
4.3.2	Simulink Blocks	11
4.3.3	Virtual Blocks	12
4.4	Subsystems	12

5	Block Compositions	12
5.1	Sequential Composition	13
5.2	Parallel Composition	14
5.3	Feedback	15
5.4	Composition Examples	17
6	Case Study	18
6.1	Modelling	18
6.2	Subsystems Verification	19
6.3	Requirement Verification	19
6.3.1	Requirement 3 and 4	20
6.3.2	Requirement 1	20
6.3.3	Requirement 2	21
6.4	Summary	21
7	Conclusions	21
7.1	Progress Summary	21
A	Block Theories	23
A.1	Additional Laws	23
A.2	State Space	24
A.3	Patterns	24
A.4	Number of Inputs and Outputs	24
A.5	Operators	25
A.5.1	Id	25
A.5.2	Parallel Composition	25
A.5.3	Sequential Composition	26
A.5.4	Feedback	26
A.5.5	Split	28
A.6	Blocks	28
A.6.1	Source	28
A.6.1.1	Constant	28
A.6.2	Unit Delay	28
A.6.3	Discrete-Time Integrator	28
A.6.4	Sum	29
A.6.5	Product	30
A.6.6	Gain	31
A.6.7	Saturation	31
A.6.8	MinMax	31
A.6.9	Rounding	32
A.6.10	Logic Operators	32
A.6.10.1	AND	33
A.6.10.2	OR	33
A.6.10.3	NAND	33
A.6.10.4	NOR	33
A.6.10.5	XOR	33
A.6.10.6	NXOR	34
A.6.10.7	NOT	34
A.6.11	Relational Operator	34
A.6.11.1	Equal ==	34

A.6.11.2	Notequal =	34
A.6.11.3	Less Than <	34
A.6.11.4	Less Than or Equal to <=	34
A.6.11.5	Greater Than >	35
A.6.11.6	Greater Than or Equal to >=	35
A.6.12	Switch	35
A.6.13	Data Type Conversion	35
A.6.14	Initial Condition (IC)	38
A.6.15	Router Block	38
B	Block Laws	38
B.1	Additional Laws	38
B.2	SimBlock healthiness	40
B.3	inps and outps	42
B.4	Operators	46
B.4.1	Id	46
B.4.2	Sequential Composition	46
B.4.3	Parallel Composition	48
B.4.3.1	<i>mergeB</i>	48
B.4.3.2	<i>sim-parallel</i>	49
B.4.4	Feedback	67
B.4.4.1	<i>feedback</i>	67
B.4.5	Split	80
B.5	Blocks	80
B.5.1	Source	80
B.5.1.1	Const	80
B.5.1.2	Pulse Generator	80
B.5.2	Unit Delay	80
B.5.3	Discrete-Time Integrator	80
B.5.4	Sum	80
B.5.5	Product	81
B.5.6	Gain	81
B.5.7	Saturation	81
B.5.8	MinMax	82
B.5.9	Rounding	82
B.5.10	Combinatorial Logic	82
B.5.11	Logic Operators	82
B.5.11.1	AND	82
B.5.11.2	OR	83
B.5.11.3	NAND	84
B.5.11.4	NOR	84
B.5.11.5	XOR	84
B.5.11.6	NXOR	85
B.5.11.7	NOT	85
B.5.12	Relational Operator	85
B.5.12.1	Equal ==	85
B.5.12.2	Notequal =	85
B.5.12.3	Less Than <	85
B.5.12.4	Less Than or Equal to <=	85

B.5.12.5 Greater Than $>$	85
B.5.12.6 Greater Than or Equal to \geq	86
B.5.13 Switch	86
B.5.14 Merge	87
B.5.15 Subsystem	87
B.5.16 Enabled Subsystem	87
B.5.17 Triggered Subsystem	87
B.5.18 Enabled and Triggered Subsystem	87
B.5.19 Data Type Conversion	87
B.5.20 Initial Condition (IC)	87
B.5.21 Router Block	87
B.6 Frequently Used Composition of Blocks	88
C Post Landing Finalize	89
C.1 Subsystem: <i>variableTimer</i>	89
C.1.1 Verification	112
C.2 Subsystem: <i>rise1Shot</i>	115
C.2.1 Verification	117
C.3 Subsystem: Latch	117
C.3.1 Verification	127
C.4 System: <i>post-landing-finalize</i>	128
C.5 Verification	169
C.5.1 Requirement 01	169
C.5.2 Requirement 02	195
C.5.3 Requirement 03	214
C.5.4 Requirement 04	215

1 Introduction

Control law diagrams such as Simulink [1] and OpenModelica [2] are widely used industrial languages and tool-sets for expressing control laws, including support for simulation and code generation. In particular, Simulink actually is a *de facto* standard in many areas in industry. Its model based design, simulation and code generation make it a very efficient and cost-effective way to develop complex systems. Though empirical analysis through simulation is an important technique to explore and refine models, only formal verification can make specific mathematical guarantees about behaviour, which is crucial to ensure safety of associated implementations. Whilst verification facilities for Simulink exist [3, 4, 5, 6, 7, 8], there is still a need for assertional reasoning techniques that capture the full range of specifiable behaviour, provide non-deterministic specification constructs, and support compositional verification. Such techniques also need to be sufficiently expressive to handle the plethora of additional languages and modelling notations that are used by industry in concert with Simulink, in order to allow formulation of heterogeneous "multi-models" that capture the different paradigms and disciplines used in large scale systems [9]. Applicable tool support for these techniques with a high degree of automation is also of vital importance to enable adoption by industry. Since Simulink diagrams are data rich and usually have an uncountably infinite state space, model checking alone is insufficient and there is a need for theorem proving facilities.

Assume-Guarantee (AG) reasoning is a valuable compositional verification technique for reactive systems [10, 11, 12]. In AG, one demonstrates composite system level properties by decompos-

ing them into a number of contracts for each component subsystem. Each contract specifies the guarantees that the subsystem will make about its behaviour, under certain specified assumptions of the subsystem’s environment. Such a decomposition is vital in order to make verification of a complex system tractable, and to allow development of subsystems by separate teams. AG reasoning has previously been applied to verification of discrete time Simulink control law diagrams through mappings into synchronous languages like Lustre [13] and Kahn Process Networks [5]. However such formalisms, whilst theoretically and practically appealing, are limited to expressing processes that are inherently deterministic and non-terminating in nature. Refinement Calculus for Reactive Systems (RCRS) [8] is a methodology that can be applied to reason about non-deterministic and non-input-receptive systems by treating programs as predicate transformers. However, it is not able to reason about multi-rate Simulink diagrams and algebraic loops. Almost all these verification facilities translate Simulink to sequential languages, synchronous languages or reactive languages [7], and then use verification methods for these languages to reason about Simulink diagrams. There is a need to develop a reasoning technique that is based on the semantic understanding of simulation in Simulink as described in Section 2.1. Thus, it is necessary to translate to several additional notations where AG verification can be performed, which hampers both traceability and composition with other languages of different paradigms. What is needed is a rich unified language capable of AG reasoning, and supported by theorem proving, into which Simulink and associated notations can be losslessly translated.

Our proposed approach thus explores development of formal AG-based proof support for discrete-time Simulink diagrams through a semantic embedding of the theory of designs [14] in Unifying Theories of Programming (UTP) [15] in Isabelle/HOL [16] using our developed tool Isabelle/UTP [17]. Initially, we proposed to use *Circus* [18], a formal modelling language for concurrent and reactive systems in the style of CSP, to model Simulink diagrams as shown in [7], and then apply contract-based reasoning to *Circus*. A *Circus* model consists of a network of processes that communicate with one another solely via shared channels that carry typed data. Internal state variables are encapsulated and not directly observable by other parallel processes. *Circus* can capture a variety of languages at the semantic level, and thus supports the formulation of heterogeneous multi-models [9] by acting as a “lingua franca”. In addition, a timed version of *Circus* is used to model multi-rate diagrams. However, a *Circus* model has more complex information of blocks in Simulink for AG reasoning. For example, the corresponding *Circus* process for a block uses channels to model connections in diagrams, a non-deterministic internal choice of all input channels to allow an arbitrary input order, and similarly an internal choice of output channels to allow an arbitrary output order.

In order to reason about the *Circus* model, we need to take trace information into account and traces inevitably are more complicated if there are many inputs and outputs for a block. Eventually, using model checking or theorem proving to verify *Circus* models becomes more difficult. According to the semantic understanding of simulation in Simulink in Section 2.1, actually the order of inputs and outputs is irrelevant. Therefore, we have changed our approach to use the theory of designs in UTP to enable AG reasoning for Simulink block diagrams.

A *design* in UTP is a relation between two predicates where the first predicate (precondition) records the assumption and the second one (postcondition) specifies the commitment. *Designs* are intrinsically suitable for modelling and reasoning about state-based programs (such as B machines [19] and Z notations [20]) but not necessary for reactive programs. For simulation of Simulink diagrams, we discretise the simulation time and abstract it into steps (natural numbers), and define inputs and outputs of Simulink blocks as a function from step numbers to a list of inputs or outputs. In this way, the reactive behaviour is encoded in the step numbers

in functions. Finally, the theory of designs can be used to reason about reactive behaviour of Simulink diagrams without introduction of detailed implementation information .

Our work presented in this report has multiple contributions. The main contribution is to define a theoretical reasoning framework for control law block diagrams using the theory of designs in UTP. Each block or subsystem is translated to a design and then hierarchical connections of blocks are mapped to a variety of compositions of designs. Additionally, the refinement relation of designs, monotony of composition operators, and closure laws enable compositional reasoning of block diagrams using a contract-based methodology. The second contribution is our mechanisation of theories in the theorem prover Isabelle using our implementation of UTP, Isabelle/UTP. Then the practical contribution is our industrial case study of a subsystem in a safety critical aircraft cabin pressure control system.

In the next section, we describe the relevant preliminary background about Simulink and UTP. Then in Section 3, the assumptions we made are presented and a brief reasoning procedure is described. Section 4 defines our treatment of blocks in UTP and translations of a number of blocks are illustrated. Furthermore, we introduce our composition operators and their corresponding theorems in Section 5. Afterwards, in Section 6 we briefly describe the industrial case study. And we conclude our work in Section 7. Additionally, our mechanised theories, laws and case studies are attached in appendices.

2 Preliminaries

2.1 Control Law Diagrams and Simulink

Simulink is a model-based design modelling, analysis and simulation tool for signal processing systems and control systems. It offers a graphical modelling language which is based on hierarchical block diagrams. Its diagrams are composed of subsystems and blocks as well as connections between these subsystems and blocks. In addition, subsystems also can consists of others subsystems and blocks. And single function blocks have inputs and outputs, and some blocks also have internal states.

There is no formal semantics for Simulink. A consistent understanding [21, 22] of the simulation in Simulink is based on an *idealized* time model. All executions and updates of blocks are performed *instantaneously* (and infinitely fast) at exact simulation steps. Between the simulation steps, the system is *quiescent* and all values held on lines and blocks are constant. The inputs, states and outputs of a block can only be updated when there is a time hit for this block. Otherwise, all values held in the block are constant too though at exact simulation steps. According to this idealized time model, it is inappropriate to assume that blocks are sequentially executed. For example, for a block it is inappropriate to say it takes its inputs, calculates its outputs and states, and then outputs the results from this point of view. Simulation and code generation of Simulink diagrams use sequential semantics for implementation. But it is not always necessary. Simulink needs to have a mathematical and denotational semantics, which UTP provides.

Based on the idealized time model, a single function block can be regarded as a relation between its inputs and outputs. For instance, a unit delay block specifies that its initial output is equal to its initial condition and its subsequent output is equal to previous input. Then connections of blocks establish further relations between blocks. A directed connection from one block to another block specifies that the output of one block is equal to the input of another block. Finally, hierarchical block diagrams establish a relation network between blocks and subsystems.

2.2 Unifying Theories of Programming

UTP is a unified framework to provide a theoretical basis for describing and specifying computer languages across different paradigms such as imperative, functional, declarative, nondeterministic, concurrent, reactive and high-order. A theory in UTP is described using three parts: *alphabet*, a set of variable names for the theory to be studied; *signature*, rules of primitive statements of the theory and how to combine them together to get more complex program; and *healthiness conditions*, a set of mathematically provable laws or equations to characterise the theory.

The alphabetised relational calculus [23] is the most basic theory in UTP. A relation is defined as a predicate with undecorated variables (v) and decorated variables (v') in its alphabet. v denotes the observation made initially and v' denotes the observation made at the intermediate or final state.

The understanding of the simulation in Simulink is very similar to the concept “programs-as-predicates” [24]. This is the similar idea that the Refinement Calculus of Reactive Systems (RCRS) [8] uses to reason about reactive systems. RCRS is a compositional formal reasoning framework for reactive systems. The language is based on monotonic property transformers which is an extension of monotonic predicate transformers [25]. This semantic understanding makes Unifying Theories of Programming (UTP) [15] intrinsically suitable for reasoning of the semantics of Simulink simulation because UTP uses an alphabetised predicate calculus to model computations.

Refinement calculus is an important concept in UTP. Program correctness is denoted by $S \sqsubseteq P$, which means that the observations of the program P must be a subset of the observations permitted by the specification S . For instance, $(x = 2)$ is a refinement of the predicate $(x > 1)$. A refinement sequence is shown in (1). $S1$ is more general and abstract specification than $S2$ and thus more easier to implement. The predicate *true* is the easiest one and can be implemented by anything. $P2$ is more specific and determinate program than $P1$ and thus $P2$ is more useful in general. *false* is the strongest predicate and it is impossible to implement in practice.

$$\mathbf{true} \sqsubseteq S1 \sqsubseteq S2 \sqsubseteq P1 \sqsubseteq P2 \sqsubseteq \mathbf{false} \quad (1)$$

2.2.1 Designs

Designs are a subset of the alphabetised predicates that use a particular variable ok to record information about the start and termination of programs. The behaviour of a design is described from initial observation and final observation by relating its precondition P (assumption) to the postcondition Q (commitment) as $P \vdash Q$ [14, 15] (assuming P holds initially, then Q is established). Therefore, the theory of designs is intrinsically suitable for assume-guarantee reasoning [26].

Definition 2.1 (Design)

$$P \vdash Q \triangleq P \wedge ok \Rightarrow Q \wedge ok'$$

A design is defined in 2.1 where ok records the program has started and ok' that it has terminated. It states that if the design has started ($ok = true$) in a state satisfying its precondition P , then it will terminate ($ok' = true$) with its postcondition Q established. We introduce some basic designs.

Definition 2.2 (Basic Designs)

$$\begin{aligned}
\top_D &\triangleq \mathbf{true} \vdash \mathbf{false} = \neg \text{ok} && \text{[Miracle]} \\
\perp_D &\triangleq \mathbf{false} \vdash \mathbf{false} = \mathbf{true} && \text{[Abort]} \\
(x := e) &\triangleq (\mathbf{true} \vdash x' = e \wedge y' = y \wedge \dots) && \text{[Assignment]} \\
\mathbb{I}_D &\triangleq (\mathbf{true} \vdash \mathbb{I}) && \text{[Skip]}
\end{aligned}$$

Abort (\perp_D) and miracle (\top_D) are the top and bottom element of a complete lattice formed from designs under the refinement ordering. Abort (\perp_D) is never guaranteed to terminate and miracle establishes the impossible. In addition, abort is refined by any other design and miracle refines any other designs. Assignment has precondition **true** provided the expression e is well-defined and establishes that only the variable x is changed to the value of e and other variables have not changed. The skip \mathbb{I}_D is a design identity that always terminates and leaves all variables unchanged.

Designs can be sequentially composed with the following theorem:

Theorem 2.1 (Sequential Composition)

$$(p_1 \vdash Q_1 ; P_2 \vdash Q_2) = ((p_1 \wedge \neg (Q_1 ; \neg P_2)) \vdash Q_1 ; Q_2) \quad \text{[}p_1\text{-condition]}$$

A sequence of designs terminates when p_1 holds and Q_1 guarantees to establish P_2 provided p_1 is a condition. On termination, sequential composition of their postconditions is established. A condition is a particular predicate that only has input variables in its alphabet. In other words, a design of which its precondition is a condition only makes the assumption about its initial observation (input variables) and without output variables. That is the same case for our treatment of Simulink blocks. Furthermore, sequential composition has two important properties: associativity and monotonicity which are given in the theorem below.

Theorem 2.2 (Associativity, Monotonicity)

$$\begin{aligned}
P_1 ; (P_2 ; P_3) &= (P_1 ; P_2) ; P_3 && \text{[Associativity]} \\
(P_1 ; Q_1) &\sqsubseteq (P_2 ; Q_2) && \text{[Monotonicity]}
\end{aligned}$$

Refinement of designs is given in the theorem below.

Theorem 2.3 (Refinement)

$$\begin{aligned}
(P_1 \vdash Q_1 \sqsubseteq P_2 \vdash Q_2) &= (P_2 \sqsubseteq P_1) \wedge (Q_1 \sqsubseteq P_1 \wedge Q_2) \\
&= [P_1 \Rightarrow P_2] \wedge [P_1 \wedge Q_2 \Rightarrow Q_1]
\end{aligned}$$

Refinement of designs is achieved by either weakening the precondition, or strengthening the postcondition in the presence of the precondition.

In addition, we define two notations pre_D and $post_D$ to retrieve the precondition of the design and the postcondition in the presence of the precondition.

Definition 2.3 (pre_D and $post_D$)

$$\begin{aligned}
pre_D(P \vdash Q) &\triangleq P \\
post_D(P \vdash Q) &\triangleq (P \Rightarrow Q)
\end{aligned}$$

3 Assumptions and General Procedure of Reasoning

3.1 Assumptions

Causality We assume the discrete-time systems modelled in Simulink diagrams are *causal* where the output at any time only depends on values of present and past inputs. Consequently, if inputs to a casual system are identical up to some time, their corresponding outputs must also be equal up to this time.

Single-rate This mechanised work captures only single sampling rate Simulink models, which means the timestamps of all simulation steps are multiples of a base period T . Eventually, steps are abstracted and measured by step numbers (natural numbers \mathbb{N}) and T is removed from its timestamp.

An *algebraic loop* occurs in simulation when there exists a signal loop with only direct feedthrough blocks in the loop, such as instantaneous feedback without delay in the loop. [5, 6, 27] assume there are no algebraic loops in Simulink diagrams and RCRS [8] identifies it as a future work. Our theoretical framework can reason about discrete-time block diagrams with algebraic loops: specifically check if there are solutions and find the solutions.

The signals in Simulink can have many data types, such as signed or unsigned integer, single float, double float, and boolean. The default type for signals are *double* in Simulink. This work uses real numbers in Isabelle/HOL as a universal type for all signals. Real numbers in Isabelle/HOL are modelled precisely using Cauchy sequences, which enables us to reason in the theorem prover. This is a reasonable simplification because all other types could be expressed using real numbers, such as boolean as 0 and 1.

3.2 General Procedure of Applying Assumption-Guarantee Reasoning

Simulink blocks are semantically mapped to designs in UTP where additionally we model assumptions of blocks to avoid unpredictable behaviour (such as a divide by zero error in the Divide block) and ensure healthiness of blocks. The general procedure of applying AG reasoning to Simulink blocks is given below.

- Single blocks and atomic subsystems are translated to single designs with assumptions and guarantees, as well as block parameters. This is shown in Section 4.
- Hierarchical block connections are modelled as compositions of designs (I) by means of sequential composition, parallel composition and feedback.
- Properties or Requirements of block diagrams (S) to be verified are modelled as designs as well.
- The refinement relation ($S \sqsubseteq I$) in UTP is used to verify if a given property is satisfied by a block diagram (or a subsystem) or not. Our approach supports compositional reasoning according to monotonicity of composition operators in terms of the refinement relation. Provided two properties S_1 and S_2 are verified to hold in two blocks or subsystems I_1 and I_2 respectively, then composition of the properties is satisfied by the composition of the blocks or subsystems in terms of the same operator.

$$(S_1 \sqsubseteq I_1 \wedge S_2 \sqsubseteq I_2) \Rightarrow (S_1 \text{ op } S_2 \sqsubseteq I_1 \text{ op } I_2)$$

4 Semantic Translation of Blocks

In this section, we focus on the methodology to map individual Simulink blocks to designs in UTP semantically. Basically, a block or subsystem is regarded as a relation between inputs and outputs. We use an undashed variable and a dashed variable to denote input signals and output signals respectively.

4.1 State Space

The state space of our theory for block diagrams is composed of only one variable in addition to ok , named $inouts$. Originally, we defined it as a function from real numbers (time t) to a list of inputs or outputs. Each element in the list denotes an input or output and their order in the list is the order of input or output signals.

$$inouts : \mathbb{R}_{\geq 0} \rightarrow \text{seq } \mathbb{R}$$

However, according to our single-rate assumption, the timestamp at time t is equal to multiples of a basic period T : $inouts(t) = inouts(n * T)$. Then T is abstracted away and only the step number n is related. Finally, it is defined below.

$$inouts : \mathbb{N} \rightarrow \text{seq } \mathbb{R}$$

Then a block is a design that establishes the relation between an initial observation $inouts$ (a list of input signals) and a final observation $inouts'$ (a list of output signals). Additionally, this is subject to the assumption of the design.

4.2 Healthiness Condition: **SimBlock**

This healthiness condition characterises a block with a fixed number of inputs and outputs. Additionally it is feasible. A design is a feasible block if there exists at least a pair of $inouts$ and $inouts'$ that establishes both the precondition and postcondition of the design.

Definition 4.1 (SimBlock**)** A design P with m inputs and n outputs is a Simulink block if P is **SimBlock** healthy.

$$\mathbf{SimBlock}(m, n, P) \triangleq \left(\begin{array}{l} (pre_D(P) \wedge post_D(P) \neq \mathbf{false}) \wedge \\ ((\forall n \bullet \#(inouts \ n) = m) \sqsubseteq Dom \ (pre_D(P) \wedge post_D(P))) \\ ((\forall n \bullet \#(inouts \ n) = n) \sqsubseteq Ran \ (pre_D(P) \wedge post_D(P))) \end{array} \right)$$

where Dom and Ran calculate the characteristic predicate for domain and range. Their definitions are shown below.

$$\begin{aligned} Dom(P) &\triangleq (\exists inouts' \bullet P) \\ Ran(P) &\triangleq (\exists inouts \bullet P) \end{aligned}$$

$inps$ and $outps$ are the operators to get the number of input signals and output signals for a block. They are implied from **SimBlock** of the block.

Definition 4.2 ($inps$ and $outps$)

$$\mathbf{SimBlock}(m, n, P) \Rightarrow (inps(P) = m \wedge outps(P) = n)$$

Provided that P is a healthy block, $inps$ returns the number of its inputs and $outps$ returns the number of its outputs.

4.3 Blocks

In order to give definitions of the corresponding designs for Simulink blocks, firstly we define a design pattern *FBlock*. Then we illustrate definitions of two typical Simulink blocks and three additional virtual blocks using this pattern. The definitions of all other blocks could be found in Appendix A.

4.3.1 Pattern

We defined a pattern that is used to define all other blocks.

Definition 4.3 (*FBlock*)

$$FBlock(f_1, m, n, f_2) \triangleq \left(\begin{array}{l} \forall nn \bullet f_1(inouts, nn) \\ \vdash \\ \forall nn \bullet \left(\begin{array}{l} \#(inouts(nn)) = m \wedge \\ \#(inouts'(nn)) = n \wedge \\ (inouts'(nn) = f_2(inouts'(nn), nn)) \wedge \\ (\forall sigs : \mathbb{N} \rightarrow \text{seq } \mathbb{R}, nn : \mathbb{N} \bullet \#(sigs\ nn) = m \Rightarrow \#(f_2(sigs, nn)) = n) \end{array} \right) \end{array} \right)$$

FBlock has four parameters: f_1 is a predicate that specifies the assumption of the block and it is a function on input signals; m and n are the number of inputs and outputs, and f_2 is a function that relates inputs to outputs and is used to establish the postcondition of the block. The precondition of *FBlock* states that f_1 holds for inputs at any step nn . And the postcondition specifies that for any step nn the block always has m inputs and n outputs, the relation between outputs and inputs are given by f_2 , and additionally f_2 always produces n outputs provided there are m inputs.

4.3.2 Simulink Blocks

Definition 4.4 (Unit Delay)

$$UnitDelay(x_0) \triangleq FBlock(true_f, 1, 1, (\lambda x, n \bullet \langle x_0 \triangleleft n = 0 \triangleright hd(x\ (n-1)) \rangle))$$

where hd is an operator to get the head of a sequence, and $true_f = (\lambda x, n \bullet true)$ that means no constraints on input signals.

The definition 4.4 of the Unit Delay block is straightforward: it accepts all inputs, has one input and one output, and produces initial value x_0 in its first step (0) and the previous input otherwise.

Definition 4.5 (Product (Divide))

$$Div2 \triangleq FBlock((\lambda x, n \bullet hd(tl(x\ n)) \neq 0), 2, 1, (\lambda x, n \bullet \langle hd(x\ n)/hd(tl(x\ n)) \rangle))$$

where tl is an operator to get the tail of a sequence.

The definition 4.5 of Divide block is slightly different because it assumes the input value of its second input signal is not zero at any step. By this way, the precondition enables modelling of non-input-receptive systems that may reject some inputs at some points.

4.3.3 Virtual Blocks

In addition to Simulink blocks, we have introduced three blocks for the purpose of composition: *Id*, *Split2*, and *Router*. The usage of these blocks is illustrated in Figure 1.

Definition 4.6 (Id)

$$Id \triangleq FBlock (true_f, 1, 1, (\lambda x, n \bullet \langle hd (x \ n) \rangle))$$

The identity block *Id* is a block that has one input and one output, and the output value is always equal to the input value. It establishes a fact that a direct signal line in Simulink could be treated as sequential composition of many *Id* blocks. The usage of *Id* is shown in Figure 1a.

Definition 4.7 (Split2)

$$Split2 \triangleq FBlock (true_f, 1, 2, (\lambda x, n \bullet \langle hd (x \ n), hd (x \ n) \rangle))$$

Split2 corresponds to the signal connection splitter that produces two signals from one and both signals are equal to the input signal. The usage of *Split2* is shown in Figure 1b.

Definition 4.8 (Router)

$$Router (m, table) \triangleq FBlock (true_f, m, m, (\lambda x, n \bullet reorder ((x \ n), table)))$$

Router corresponds to the crossing connection of signals and this virtual block changes the order of input and output signals according to the supplied table. The usage of *Router* is shown in Figure 1c.

4.4 Subsystems

The treatment of subsystems (no matter whether hierarchical subsystems or atomic subsystems) in our designs is similar to that of blocks. They could be regarded as a bigger black box that relates inputs to outputs.

5 Block Compositions

In this section, we define three composition operators that are used to compose subsystems or systems from blocks. We also use three virtual blocks to map Simulink's connections in our designs.

For all definitions and laws in this section, if there are no special notes, we assume the following predicates.

$$\mathbf{SimBlock} (m_1, n_1, P_1)$$

$$\mathbf{SimBlock} (m_2, n_2, P_2)$$

$$\mathbf{SimBlock} (m_3, n_3, P_3)$$

$$\mathbf{SimBlock} (m_1, n_1, Q_1)$$

$$\mathbf{SimBlock} (m_2, n_2, Q_2)$$

$$P_1 \sqsubseteq Q_1$$

$$P_2 \sqsubseteq Q_2$$

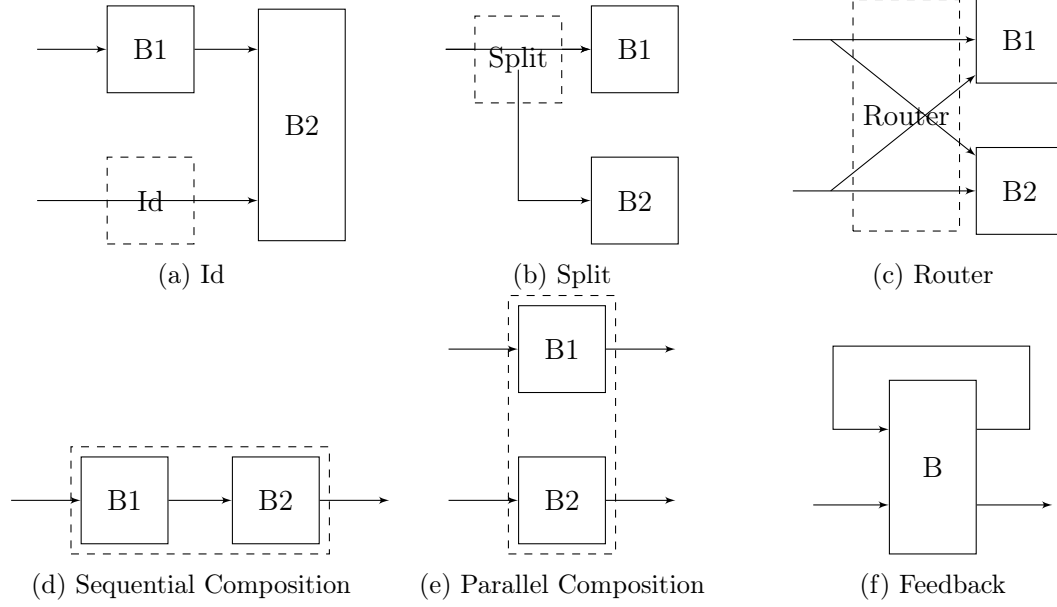


Figure 1: Composition of Blocks

5.1 Sequential Composition

The meaning of sequential composition of designs is defined in Theorem 2.1. It corresponds to composition of two blocks in Figure 1d where the outputs of B_1 are equal to the inputs of B_2 . Provided

$$\begin{aligned}
 P &= (FBlock(true_f, m_1, n_1, f_1)) & \mathbf{SimBlock}(m_1, n_1, P) \\
 Q &= (FBlock(true_f, n_1, n_2, f_2)) & \mathbf{SimBlock}(n_1, n_2, Q)
 \end{aligned}$$

The expansion law of sequential composition is given below.

Theorem 5.1 (Expansion)

$$(P; Q) = FBlock(true_f, m_1, n_2, (f_2 \circ f_1)) \quad [\text{Expansion}]$$

This theorem establishes that sequential composition of two blocks, where the number of outputs of the first block is equal to the number of inputs of the second block, is simply a new block with the same number of inputs as the first block P and the same number of outputs as the second block Q , and additionally the postcondition of this composed block is function composition. In addition, the composed block is still **SimBlock** healthy which is shown in the closure theorem below.

Theorem 5.2 (Closure)

$$\mathbf{SimBlock}(m_1, n_2, (P; Q)) \quad [\mathbf{SimBlock} \text{ Closure}]$$

5.2 Parallel Composition

Parallel composition of two blocks is a stack of inputs and outputs from both blocks and is illustrated in Figure 1e. It is defined below.

Definition 5.1 (Parallel Composition)

$$P \parallel_B Q \triangleq \left(\begin{array}{l} (takem(inps(P) + inps(Q)) \ inps(P); \ P) \\ \parallel_{B_M} \\ (dropm(inps(P) + inps(Q)) \ inps(P); \ Q) \end{array} \right)$$

where *takem* and *dropm* are two blocks to split inputs into two parts and their definitions can be found in Appendix A, and B_M is defined below.

Definition 5.2 (B_M)

$$B_M \triangleq (ok' = 0.ok \wedge 1.ok) \wedge (inouts' = 0.inouts \hat{\ } 1.inouts)$$

The definition of parallel composition 5.1 for designs is similar to the parallel-by-merge scheme [15, Sect. 7.2] in UTP. Parallel-by-merge is denoted as $P \parallel_M Q$ where M is a special relation that explains how the output of parallel composition of P and Q should be merged following execution.

However, parallel-by-merge assumes that the initial observations for both predicates should be the same. But that is not the case for our block composition because the inputs to the first block and that to the second block are different. Therefore, in order to use the parallel by merge, firstly we need to partition the inputs to the composition into two parts: one to the first block and another to the second block. This is illustrated in Figure 2 where we assume that P has m inputs and i outputs, and Q has n inputs and j outputs. Finally, it has the same inputs ($m + n$) and the outputs of P and Q are merged by B_M to get $i + j$ outputs.

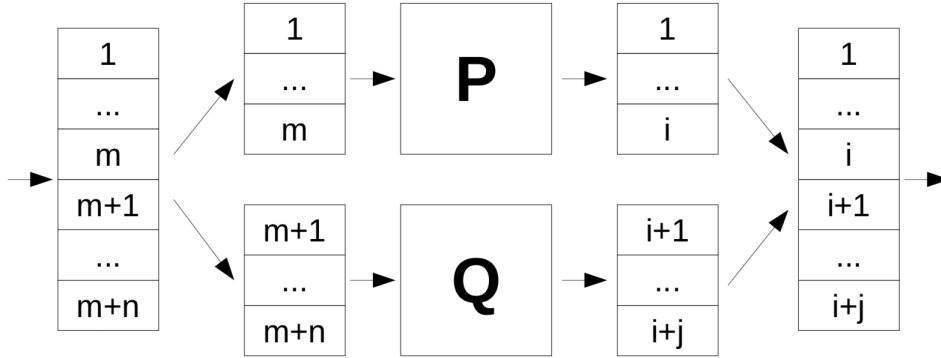


Figure 2: Parallel Composition of Two Blocks

The merge operator B_M states that the parallel composition terminates if both blocks terminate. And on termination, the output of parallel composition is concatenation of the outputs from the first block and the outputs from the second block. *takem* and *dropm* are two blocks that have the same inputs and the number of inputs is equal to addition of the number inputs of P and the number inputs of Q . *takem* only takes the first part of inputs as required by P , and *dropm* takes the second part of inputs as required by Q .

Theorem 5.3 (Associativity, Monotonicity, and *SimBlock* Closure)

$$\begin{aligned}
P_1 \parallel_B (P_2 \parallel_B P_3) &= (P_1 \parallel_B P_2) \parallel_B P_3 && \text{[Associativity]} \\
(P_1 \parallel_B Q_1) \sqsubseteq (P_2 \parallel_B Q_2) &&& \text{[Monotonicity]} \\
\mathbf{SimBlock}(m_1 + m_2, n_1 + n_2, (P_1 \parallel_B P_2)) &&& \text{[SimBlock Closure]} \\
\text{inps}(P_1 \parallel_B P_2) &= m_1 + m_2 \\
\text{outps}(P_1 \parallel_B P_2) &= n_1 + n_2
\end{aligned}$$

Parallel composition is associative, monotonic in terms of the refinement relation, and *SimBlock* healthy. The inputs and outputs of parallel composition are combination of the inputs and outputs of both blocks.

Theorem 5.4 (Parallel Operator Expansion) *Provided*

$$\begin{aligned}
P &= (FBlock(\text{true}_f, m_1, n_1, f_1)) && \mathbf{SimBlock}(m_1, n_1, P) \\
Q &= (FBlock(\text{true}_f, m_2, n_2, f_2)) && \mathbf{SimBlock}(m_2, n_2, Q)
\end{aligned}$$

then,

$$\begin{aligned}
(P \parallel_B Q) &= FBlock \left(\text{true}_f, m_1 + m_2, n_1 + n_2, \left(\lambda x, n \bullet \left(\begin{array}{l} (f_1 \circ (\lambda x, n \bullet \text{take}(m_1, x \ n))) \\ \wedge (f_2 \circ (\lambda x, n \bullet \text{drop}(m_1, x \ n))) \end{array} \right) \right) \right) && \text{[Expansion]} \\
&\mathbf{SimBlock}(m_1 + m_2, n_1 + n_2, (P \parallel_B Q)) && \text{[SimBlock Closure]}
\end{aligned}$$

Parallel composition of two *FBlock* defined blocks is expanded to get a new block. Its postcondition is concatenation of the outputs from P and the outputs from Q . The outputs from P (or Q) are function composition of its block definition function f_1 (or f_2) with *take* (or *drop*).

5.3 Feedback

The feedback operator loops an output back to an input, which is illustrated in Figure 1f.

Definition 5.3 (f_D)

$$P \ f_D \ (i, o) \triangleq (\exists \text{sig} \bullet (\text{PreFD}(\text{sig}, \text{inps}(P), i); P; \text{PostFD}(\text{sig}, \text{outps}(P), o)))$$

where i and o denotes the index number of the output signal and the input signal, which are looped. *PreFD* denotes a block that adds *sig* into the i th place of the inputs.

Definition 5.4 (*PreFD*)

$$\text{PreFD}(\text{sig}, m, idx) \triangleq FBlock(\text{true}_f, m - 1, m, (f_PreFD(\text{sig}, idx)))$$

where $f_PreFD(\text{sig}, idx) = \lambda x, n \bullet (\text{take}(idx, (x \ n)) \wedge \langle \langle \text{sig} \ n \rangle \rangle \wedge \text{drop}(idx, (x \ n)))$

and *PostFD* denotes a block that removes the o th signal from the outputs of P and this signal shall be equal to *sig*.

Definition 5.5 (*PostFD*)

$$PostFD(sig, n, idx) \triangleq \left(\begin{array}{l} \mathbf{true} \\ \vdash \\ \forall nn \bullet \left(\begin{array}{l} \#(inouts(nn)) = n \wedge \\ \#(inouts'(nn)) = n - 1 \wedge \\ (inouts'(nn) = (f_PostFD(sig, idx, inouts'(nn), nn)) \wedge \\ sig(nn) = inouts(nn)!idx \end{array} \right) \end{array} \right)$$

where $f_PostFD(idx) = \lambda x, n \bullet (take(idx, (x \ n)) \hat{\ } drop(idx + 1, (x \ n)))$ and $!$ is an operator to get the element in a list by its index.

The basic idea to construct a feedback operator is to use existential quantification to specify that there exists one signal sig that it is the i th input and o th output, and their relation is established by the block P . This is illustrated in Figure 3 where m and n are the number of inputs and outputs of P . $PreFD$ adds a signal into the inputs at i and P takes assembled inputs and produces an output in which the o th output is equal to the supplied signal. Finally, the outputs of feedback are the outputs of P without the o th output. Therefore, a block with feedback is translated to a sequential composition of $PreFD$, P , and $PostFD$.

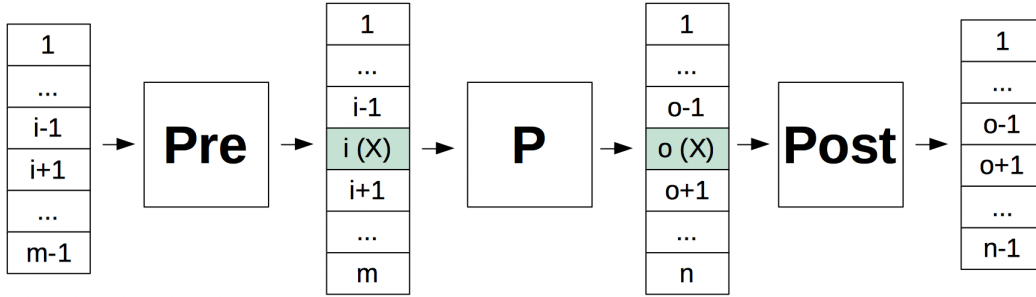


Figure 3: Feedback

Theorem 5.5 (Monotonicity) *Provided*

$$\begin{array}{ll} \mathbf{SimBlock}(m_1, n_1, P_1) & \mathbf{SimBlock}(m_1, n_1, P_2) \\ P_1 \sqsubseteq P_2 & i_1 < m_1 \wedge o_1 < n_1 \end{array}$$

then,

$$(P_1 \ f_D \ (i_1, o_1)) \sqsubseteq (P_2 \ f_D \ (i_1, o_1))$$

The monotonicity law states that if a block is a refinement of another block, then its feedback is also a refinement of the same feedback of another block.

Theorem 5.6 (Expansion) *Provided*

$$\begin{array}{ll} P = FBlock(true_f, m, n, f) & \mathbf{SimBlock}(m, n, P) \\ Solvable_unique(i, o, m, n, f) & is_Solution(i, o, m, n, f, sig) \end{array}$$

6 Case Study

This case study, verification of a `post_landing_finalize` subsystem, is taken from an aircraft cabin pressure control application. The original Simulink model is from [Honeywell](#) through our industrial link with [D-RisQ](#). This case is also studied in [28] and the diagram shown in Figure 4 is from the paper. The purpose of this subsystem is to implement that the output `finalize_event` is triggered after the aircraft door has been open for a minimum specific amount of time following a successful landing.

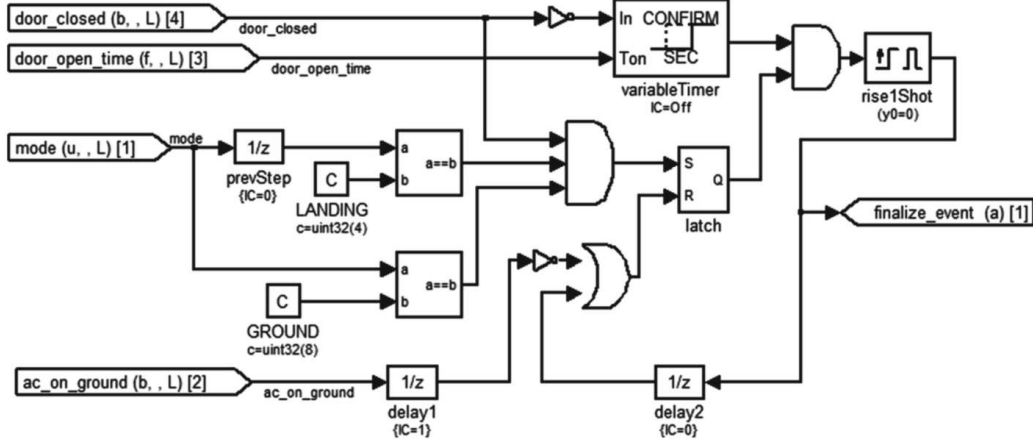


Figure 4: Post Landing Finalize (source: [28])

In order to apply our AG reasoning into this Simulink model, firstly we model the subsystem in our block theories as shown in Section 6.1. Then we verify a number of properties for three small subsystems in this model, which is given in Section 6.2. Finally, in Section 6.3 we present verification of four requirements of this subsystem. To avoid confusion between the subsystem and three small subsystems, in the following sections we use the *system* to denote the `post_landing_finalize` subsystem to be verified, and the *subsystems* to denote three small subsystems.

6.1 Modelling

We start with translation of three small subsystems (`variableTimer`, `rise1Shot` and `latch`) according to our block theories.

The subsystem `latch` is modelled as below. It is shown in Appendix C.3 as well.

$$((((UnitDelay\ 0) \parallel_B Id); (LopOR\ 2)) \parallel_B (Id; LopNOT)); (LopAND\ 2); Split2) f_D (0, 0)$$

The blocks `LopOR`, `LopNOT` and `LopAND` correspond to the OR, NOT and AND operators in the logic operator block. Their definitions can be found in Appendix A. Then we apply composition definitions, expansion and SimBlock closure laws to simplify the subsystem. The `latch` subsystem is finally simplified to a design.

$$latch = FBlock(true_f, 2, 1, latch_simp_pat_f)$$

where the definition of `latch_simp_pat_f` is given in Appendix C.

Similarly, `variableTimer` and `rise1Shot` are modelled and simplified as shown in Appendix C.1 and C.2 respectively.

Finally, we can use the similar way to compose the three subsystems with other blocks in this diagram to get the corresponding composition of `post_landing_finalise_1`, and then apply the similar laws to simplify it further into one block and verify requirements for this system. However, for the outermost feedback it is difficult to use the similar way to simplify it into one block because it is more complicated than feedbacks in other three small subsystems (`variableTimer`, `rise1Shot` and `latch`). In order to use the expansion theorem 5.6 of feedback, we need to find a solution for the block and prove the solution is unique. With increasing complexity of blocks, this expansion is becoming harder and harder. Therefore, `post_landing_finalise_1` has not been simplified into one block. Instead, it is simplified to a block with a feedback which can be seen in the lemma `post_landing_finalize_1_simp` in Appendix C.

$$post_landing_finalize_1 = plf_rise1shot_simp f_D (4, 1)$$

6.2 Subsystems Verification

After simplification, we can verify properties of the subsystems using the refinement relation.

We start with verification of a property for `variableTimer`: `vt_req_00`. This property states that if the door is closed, then the output of this subsystem is always false. The verification of this property is given in Appendix C.1.1. However, this property can not be verified in absence of an assumption made to the second input: `door_open_time`. This is due to a type conversion block `int32` used in the subsystem. If the input to `int32` is larger than 2147483647 (that is, `door_open_time` larger than 2147483647/10), its output is less than zero and finally the output is true. That is not the expected result. Practically, `door_open_time` should be less than 2147483647/10. Therefore, we can make an assumption of the input and eventually verify this property as given in the lemma `vt_req_00`. Additionally, we suggest a substitution of `int32` by `uint32`, or a change of the data type for the input from double to unsigned integer, such as `uint32`.

As for the `rise1Shot` subsystem, we verified one property: `rise1shot_req_00`. This property specifies that the output is true only when current input is true and previous input is false (see Appendix C.2.1). It means it is triggered only by a rising edge and continuous true inputs will not enable the output.

Furthermore, one property for the `latch` subsystem (a SR AND-OR latch) is verified (see Appendix C.3.1). The property `latch_req_00` states that as long as the second input `R` is true, its output is always false. This is consistent with the definition of the SR latch in circuits.

6.3 Requirement Verification

The four requirements to be verified are illustrated in Table 1.

Our approach to cope with the difficulty to simplify this system into one design is to apply compositional reasoning. Generally, application of compositional reasoning to verify requirements is as follows.

- In order to verify the property satisfied by `post_landing_finalise_1`:

$$C \sqsubseteq post_landing_finalise_1$$

, that is, to verify

$$C \sqsubseteq (plf_rise1shot_simp f_D (4, 1))$$

Requirement 1	A finalize event will be broadcast after the aircraft door has been open continuously for <i>door_open_time</i> seconds while the aircraft is on the ground after a successful landing.
Requirement 2	A finalize event is broadcast only once while the aircraft is on the ground.
Requirement 3	The finalize event will not occur during flight.
Requirement 4	The finalize event will not be enabled while the aircraft door is closed.

Table 1: Requirements for the system (source: [28])

;

- We need to find a decomposed contract C' such that

$$C \sqsubseteq (C' \text{ } f_D \text{ } (4, 1))$$

and

$$(C' \sqsubseteq \textit{plf_rise1shot_simp})$$

;

- Then we get

$$(C' \text{ } f_D \text{ } (4, 1)) \sqsubseteq (\textit{plf_rise1shot_simp} \text{ } f_D \text{ } (4, 1))$$

using the monotonicity theorem 5.5 of feedback;

- Finally, according to transitivity of the refinement relation, it establishes that

$$C \sqsubseteq (\textit{plf_rise1shot_simp} \text{ } f_D \text{ } (4, 1))$$

.

6.3.1 Requirement 3 and 4

Requirement 3 and 4 are verified together as shown in Appendix C.5.4. *req_04_contract* and *req_04_1_contract* are C and C' described above respectively.

6.3.2 Requirement 1

According to Assumption 3 “*door_open_time* does not change while the aircraft is on the ground” and the fact that this requirement specifies the aircraft is on the ground, therefore *door_open_time* is constant for this scenario. In order to simplify the verification, we assume it is always constant. The contract *req_01_contract* specifies that

- it always has four inputs and one output;
- and the requirement:
 - after a successful landing at step m and $m + 1$: the door is closed, the aircraft is on ground, and the mode is switched from LANDING (at step m) to GROUND (at step $m + 1$),

- then the door has been open continuously for $door_open_time$ seconds from step $m + 2 + p$ to $m + 2 + p + door_open_time$, therefore the door is closed at the previous step $m + 2 + p - 1$,
- while the aircraft is on ground: ac_on_ground is true and $mode$ is GROUND,
- additionally, between step m and $m + 2 + p$, the $finalize_event$ is not enabled,
- then a $finalize_event$ will be broadcast at step $m + 2 + p + door_open_time$.

As shown in Appendix C.5.1, this requirement has been verified.

6.3.3 Requirement 2

The contract $req_02_contract$ specifies that

- it always has four inputs and one output;
- and the requirement:
 - if a finalize event has been broadcast at step m ,
 - while the aircraft is on ground: ac_on_ground is true and $mode$ is GROUND,
 - then a finalize event will not be broadcast again.

As shown in Appendix C.5.2, this requirement has been verified too.

6.4 Summary

In sum, we have translated and mechanised the `post_landing_finalize` diagram in Isabelle/UTP, simplified its three subsystems (`variableTimer`, `rise1Shot` and `latch`) and the `post_landing_finalize` into a design with feedback, and finally verified all four requirements of this system. In addition, our work has identified a vulnerable block in `variableTimer`. This case study demonstrates that our verification framework has rich expressiveness to specify scenarios for requirement verification (as illustrated in the verification of Requirement 1 and 2) and our verification approach is useful in practice.

7 Conclusions

In this report, we present our work for the VeTSS funded project “Mechanised Assume-Guarantee Reasoning for Control Law Diagrams via Circus” from developed theories and laws as well as their mechanisation in Isabelle/UTP. In addition, we present practical application of our theories to reason about a Simulink model in the aircraft cabin pressure control application. Our mechanisation is also attached to this report.

7.1 Progress Summary

The project was initially proposed to have four work packages. And a summary of progress is shown in Table 2.

WP1 – framework: we reviewed current solutions that use contract-based reasoning and Circus-based program verification for Simulink. Eventually we put forward a new contract-based assume-guarantee reasoning methodology for Simulink diagrams. The theoretical part of this approach is based on the theory of design in UTP that is presented in this report.

Work Package	Description	Progress
WP1	Review current Simulink reasoning solutions and put forward a new contract-based methodology (using UTP design theory) to reason about faulty behaviour through assumptions	100%
WP2	Define assumption-guarantee contracts for the Simulink semantics and mechanise them in Isabelle/UTP, including operators and a limited selection of Simulation discrete blocks that are used in our case studies, and mechanise in Isabelle/UTP	100%
WP3	Mechanise industrial case studies (building case and post landing finalize case) in Isabelle/UTP using mechanised block libraries (produced in WP2), including modelling, contract calculation, and proof	50%
WP4	Investigate the weakest assumption calculus based on the examples, in order to automate reasoning about interferences between blocks and subsystems	25%

Table 2: Project Progress Summary

WP2 – definition and mechanisation: one advantage of using designs for reasoning is its existing theory and mechanisation in Isabelle/UTP. However, in order to accommodate Simulink diagrams into designs easily, we have defined three additional virtual blocks (Identity, Split and Router) and two extra operators (Parallel Composition and Feedback). They correspond to signal connections and block composition in Simulink. With these new blocks and operators (as well as existing operators for designs), we could translate Simulink diagrams into composition of designs. In addition, we have mechanised (in Isabelle/UTP) the three virtual blocks and 14 Simulink blocks (Constant, Unit Delay, Discrete-Time Integrator, Sum, Product, Gain, Saturation, MinMax, Rounding, Logic Operator, Relational Operator, Switch, Data Type Conversion and Initial Condition) that will be used in our case studies.

WP3 – case studies: using definitions and mechanisation of these blocks and operators, we have mechanised one of our case study (the post landing finalize) in Isabelle/UTP.

WP4 - Though time did not permit us to consider the weakest assumption calculus for Simulink in details, in a parallel project we have explored a calculus for weakest reactive rely conditions for reactive contracts based in UTP. The details of this can be found in a draft journal paper under review for Theoretical Computer Science [26]. This initial study provides necessary background for future work with Simulink.

Due to the fact that we started this project two months late since October 2017 because of delays in receiving funding, therefore we have limited time to finish all proposed work. We have not verified all requirements of the post landing finalize case, have not started the second building case study, and have investigated WP4 partially.

Acknowledgements. This project is funded by the National Cyber Security Centre (NCSC) through UK Research Institute in Verified Trustworthy Software Systems (VeTSS) [29]. We thank Honeywell and D-RisQ for sharing of the industrial case.

A Block Theories

In this section, we define main theories of block diagrams in UTP.

```
theory simu-contract-real
imports
  ~~/src/HOL/Word/Word
  utp-designs
begin
```

```
syntax
  -svid-des :: svid ( $\mathbf{v}_D$ )
```

```
translations
  -svid-des =>  $\Sigma_D$ 
```

Defined Simulink blocks using designs directly.

```
named-theorems sim-blocks
```

Functions used to define Simulink blocks via patterns.

```
named-theorems f-blocks
```

Defined Simulink blocks using functions and patterns.

```
named-theorems f-sim-blocks
```

SimBlock healthiness.

```
named-theorems simblock-healthy
```

```
recall-syntax
```

A.1 Additional Laws

```
theorem ndesign-composition:
  (( $p1 \vdash_n Q1$ ) ;; ( $p2 \vdash_n Q2$ )) = (( $p1 \wedge \neg \lfloor Q1 \rfloor$  ;; ( $\neg \lfloor p2 \rfloor \lfloor \cdot \rfloor$ ))  $\vdash_n$  ( $Q1$  ;;  $Q2$ ))
apply (ndes-simp, simp add: wp-upred-def)
by (rel-simp)
```

```
lemma list-equal-size2:
```

```
fixes  $x$ 
assumes  $\text{length}(x) = 2$ 
shows  $x = [\text{hd}(x)] \bullet [\text{last}(x)]$ 
proof -
  have  $1: x = [\text{hd}(x)] \bullet \text{tl}(x)$ 
    by (metis append-Cons append-Nil assms hd-Cons-tl length-0-conv zero-not-eq-two)
  have  $2: \text{tl}(x) = [\text{last}(x)]$ 
    using assms
    by (metis One-nat-def 1 append-butlast-last-id append-eq-append-conv append-is-Nil-conv
      cancel-ab-semigroup-add-class.add-diff-cancel-left' length-Cons length-tl list.size(3)
      nat-1-add-1 not-Cons-self2)
  from  $1$  and  $2$  show ?thesis
  by auto
qed
```


theorem *ndesign-refinement*:

$$(P1 \vdash_n Q1 \sqsubseteq P2 \vdash_n Q2) \longleftrightarrow (P1 \Rightarrow P2' \wedge \lceil P1 \rceil_{<} \wedge Q2 \Rightarrow Q1')$$

by (*rel-auto*)

theorem *ndesign-refinement'*:

$$(P1 \vdash_n Q1 \sqsubseteq P2 \vdash_n Q2) \longleftrightarrow (P2 \sqsubseteq P1 \wedge Q1 \sqsubseteq (\lceil P1 \rceil_{<} \wedge Q2))$$

by (*meson ndesign-refinement refBy-order*)

lemma *assume-Ran*: $P ; ; \text{[Ran}(P)]^\top = P$

apply (*rel-auto*)

done

fun *sum-list1* **where**

$$\text{sum-list1 } [] = 0 \mid$$

$$\text{sum-list1 } (x\#xs) = (\text{sum-list1 } xs + x)$$

A.2 State Space

inouts: input and output signals, abstracted as a function from step numbers to a list of inputs or outputs where we use universal real number as the data type of signals.

alphabet *sim-state* =

$$\text{inouts} :: \text{nat} \Rightarrow \text{real list}$$

A.3 Patterns

FBlock is a pattern to define a block with precondition, number of inputs, number of outputs, and postcondition.

definition *FBlock* ::

$$\begin{aligned} & ((\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow \text{bool}) \Rightarrow \\ & \quad \text{nat} \Rightarrow \text{nat} \Rightarrow \\ & \quad ((\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})) \Rightarrow \\ & \quad \text{sim-state hrel-des} \textbf{ where} \end{aligned}$$

[*sim-blocks*]: *FBlock* *pre m nn f* =

$$\begin{aligned} & ((\forall n::\text{nat} \cdot (\ll \text{pre} \gg (\&\text{inouts})_a (\ll n \gg)_a)::\text{sim-state upred}) \vdash_n \\ & \quad ((\forall n::\text{nat} \cdot \\ & \quad \quad ((\#_u (\$ \text{inouts } (\ll n \gg)_a)) =_u \ll m \gg) \wedge \\ & \quad \quad ((\#_u (\$ \text{inouts}' (\ll n \gg)_a)) =_u \ll nn \gg) \wedge \\ & \quad \quad (\ll f \gg (\$ \text{inouts})_a (\ll n \gg)_a =_u (\$ \text{inouts}' (\ll n \gg)_a))) \wedge \\ & \quad \quad (\forall x \cdot (\forall n::\text{nat} \cdot ((\#_u (\ll x \gg (\ll n \gg)_a)) =_u \ll m \gg) \Rightarrow (\#_u (\ll f \gg (\ll x \gg)_a (\ll n \gg)_a) =_u \ll nn \gg)))) \\ & \quad (* \text{ for any inputs, f always produces the same size output. Useful to prove FBlock-seq-comp *) \\ & \quad)) \end{aligned}$$

lemma *pre-true* [*simp*]: $(\forall n::\text{nat} \cdot (\ll \lambda x n. \text{True} \gg (\&\text{inouts})_a (\ll n \gg)_a)::\text{sim-state upred}) = \text{true}$

by (*rel-simp*)

A.4 Number of Inputs and Outputs

abbreviation *PrePost*(*P*) $\equiv \text{pre}_D(P) \wedge \text{post}_D(P)$

SimBlock is a condition stating that a design is a Simulink block if it is feasible, and has *m* inputs and *n* outputs.

definition *SimBlock* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{sim-state hrel-des} \Rightarrow \text{bool}$

where [*sim-blocks*]:

$SimBlock\ m\ n\ P = ((PrePost(P) \neq false) \wedge (*\ This\ is\ stronger\ than\ just\ excluding\ abort\ and\ miracle,\ and\ also\ not\ the\ same\ as\ H_4\ feasibility\ *))$
 $((\forall\ na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m \rangle) \sqsubseteq Dom(PrePost(P))) \wedge$
 $((\forall\ na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle n \rangle) \sqsubseteq Ran(PrePost(P)))(* \wedge$
 $(P\ is\ \mathbf{N})*))$

axiomatization

$inps :: sim\ state\ hrel\ des \Rightarrow nat\ and$

$outps :: sim\ state\ hrel\ des \Rightarrow nat$

where

$inps\ outps: (SimBlock\ m\ n\ P) \longrightarrow (inps\ P = m) \wedge (outps\ P = n)$

A.5 Operators

A.5.1 Id

definition $f\text{-}Id :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f\text{-}blocks]: f\text{-}Id\ x\ n = [hd(x\ n)]$

Id block: one input and one output, and the output is always equal to the input

definition $Id :: sim\ state\ hrel\ des$ **where**

$[f\text{-}sim\ blocks]: Id = FBlock\ (\lambda x\ n.\ True)\ 1\ 1\ (f\text{-}Id)$

A.5.2 Parallel Composition

definition $mergeB ::$

$((sim\ state\ des,\ sim\ state\ des,\ sim\ state\ des)\ mrg,$

$sim\ state\ des)\ urel\ (B_M)$ **where**

$[sim\ blocks]: mergeB = ((\$ok' =_u (\$0-ok \wedge \$1-ok)) \wedge ($

$(\forall\ n::nat \cdot ((\$v_D:inouts'(\langle n \rangle)_a) =_u (\langle append \rangle (\$0-v_D:inouts(\langle n \rangle)_a)_a (\$1-v_D:inouts(\langle n \rangle)_a)_a)$
 $(*\wedge (\#_u(\$v_D:inouts_{<}(\langle n \rangle)_a) =_u 2)*))))$

$takem$: a block that just takes the first $nr2$ inputs and ignores the remaining inputs.

definition $takem :: nat \Rightarrow nat \Rightarrow sim\ state\ hrel\ des$ **where**

$[sim\ blocks]: takem\ nr1\ nr2 = ((\langle nr2 \rangle \leq_u \langle nr1 \rangle) \vdash_n$

$(\forall\ n::nat \cdot$

$(uconj\ ((\#_u(\$inouts(\langle n \rangle)_a)) =_u \langle nr1 \rangle)$

$(uconj\ ((\#_u(\$inouts'(\langle n \rangle)_a)) =_u \langle nr2 \rangle)$

$(true \triangleleft (\langle nr2 \rangle =_u 0) \triangleright (\langle take \rangle (\langle nr2 \rangle)_a (\$inouts(\langle n \rangle)_a)_a =_u (\$inouts'(\langle n \rangle)_a)_a))$

$))))$

$dropm$: a block that just drops the first $nr2$ inputs and outputs the remaining inputs.

definition $dropm :: nat \Rightarrow nat \Rightarrow sim\ state\ hrel\ des$ **where**

$[sim\ blocks]: dropm\ nr1\ nr2 = ((\langle nr2 \rangle \leq_u \langle nr1 \rangle) \vdash_n$

$(\forall\ n::nat \cdot$

$(uconj\ ((\#_u(\$inouts(\langle n \rangle)_a)) =_u \langle nr1 \rangle)$

$(uconj\ ((\#_u(\$inouts'(\langle n \rangle)_a)) =_u \langle nr2 \rangle)$

$(true \triangleleft (\langle nr2 \rangle =_u 0) \triangleright (\langle drop \rangle (\langle nr1 - nr2 \rangle)_a (\$inouts(\langle n \rangle)_a)_a =_u (\$inouts'(\langle n \rangle)_a)_a))$

$))))$

We use the similar parallel-by-merge in UTP to implement parallel composition.

definition $sim\ parallel ::$

$sim\ state\ hrel\ des \Rightarrow$

$sim\ state\ hrel\ des \Rightarrow$

$sim\ state\ hrel\ des\ (infixl\ ||_B\ 60)$

where $[sim\text{-blocks}]$: $P \parallel_B Q =$
 $((\text{takem } (inps\ P + inps\ Q) (inps\ P)) \ ; \ ; \ P)$
 \parallel_{mergeB}
 $((\text{dropm } (inps\ P + inps\ Q) (inps\ Q)) \ ; \ ; \ Q))$

A.5.3 Sequential Composition

It is the same as the sequential composition for designs.

A.5.4 Feedback

definition $f\text{-PreFD} :: (\text{nat} \Rightarrow \text{real}) (* \text{ input signal: introduced by exists } *)$
 $\Rightarrow \text{nat} (* \text{ the input index number that is fed back from output. } *)$
 $\Rightarrow (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat}$
 $\Rightarrow \text{real list}$ **where**
 $[f\text{-blocks}]$: $f\text{-PreFD } x \text{ idx-fd } inouts0 \ n =$
 $(\text{take } \text{idx-fd } (inouts0 \ n)) \bullet (x \ n) \# (\text{drop } \text{idx-fd } (inouts0 \ n))$

definition $f\text{-PostFD} ::$
 $\text{nat} (* \text{ the input index number that is fed back from output. } *)$
 $\Rightarrow (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat}$
 $\Rightarrow \text{real list}$ **where**
 $[f\text{-blocks}]$: $f\text{-PostFD } \text{idx-fd } inouts0 \ n =$
 $(\text{take } \text{idx-fd } (inouts0 \ n)) \bullet (\text{drop } (\text{idx-fd} + 1) (inouts0 \ n))$

definition $\text{PreFD} ::$
 $(\text{nat} \Rightarrow \text{real}) (* \text{ input signal: introduced by exists } *)$
 $\Rightarrow \text{nat} (* m *)$
 $\Rightarrow \text{nat} (* \text{ the input index number that is fed back from output. } *)$
 $\Rightarrow \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{PreFD } x \ \text{nr-of-inputs } \text{idx-fd} = (\text{true} \vdash_n$
 $(\forall \ n :: \text{nat} \cdot ($
 $((\#_u (\$inouts \ (\langle n \rangle)_a)) =_u \langle \text{nr-of-inputs} - 1 \rangle) \wedge$
 $((\#_u (\$inouts' \ (\langle n \rangle)_a)) =_u \langle \text{nr-of-inputs} \rangle) \wedge$
 $(\$inouts' \ (\langle n \rangle)_a =_u (\langle f\text{-PreFD } x \ \text{idx-fd} \rangle (\$inouts)_a \ (\langle n \rangle)_a))$
 $)))$

definition $\text{PostFD} :: (\text{nat} \Rightarrow \text{real}) (* \text{ input signal: introduced by exists } *)$
 $\Rightarrow \text{nat} (* m *)$
 $\Rightarrow \text{nat} (* \text{ the input index number that is fed back from output. } *)$
 $\Rightarrow \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{PostFD } x \ \text{nr-of-inputs } \text{idx-fd} =$
 $(\text{true} \vdash_n$
 $(\forall \ n :: \text{nat} \cdot ($
 $((\#_u (\$inouts \ (\langle n \rangle)_a)) =_u \langle \text{nr-of-inputs} \rangle) \wedge$
 $((\#_u (\$inouts' \ (\langle n \rangle)_a)) =_u \langle \text{nr-of-inputs} - 1 \rangle) \wedge$
 $(\$inouts' \ (\langle n \rangle)_a =_u (\langle f\text{-PostFD } \text{idx-fd} \rangle (\$inouts)_a \ (\langle n \rangle)_a)) \wedge$
 $((\langle nth \rangle (\$inouts \ (\langle n \rangle)_a) \ (\langle \text{idx-fd} \rangle)_a =_u \langle x \ n \rangle))$
 $)))$

The feedback operator *sim-feedback* is defined via existential quantification.

fun $\text{sim-feedback} :: \text{sim-state hrel-des}$

$\Rightarrow (\text{nat} * \text{nat})$
 $\Rightarrow \text{sim-state hrel-des (infixl } f_D \text{ 60)}$

where

$P f_D (i1, o1) = (\exists (x) \cdot (\text{PreFD } x (\text{inps } P) i1 ;; P ;; \text{PostFD } x (\text{outps } P) o1))$

Solvable checks if the supplied function for feedback is solvable according to the feedback signal from the output $o1$ to the input $i1$. A function is solvable if its feedback is feasible. Feedback may lead to algebraic loops but this condition states that algebraic loops are solvable.

definition *Solvable*:: $\text{nat} (* \text{ the input index for feedback } *)$

$\Rightarrow \text{nat} (* \text{ the output index for feedback } *)$
 $\Rightarrow \text{nat} (* \text{ how many input signals } *)$
 $\Rightarrow \text{nat} (* \text{ how many output signals } *)$
 $\Rightarrow ((\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow \text{real list}) (* \text{ function } *)$
 $\Rightarrow \text{bool}$ **where**

$\text{Solvable } i1 \ o1 \ m \ nn \ f = ((i1 < m \wedge o1 < nn) \wedge$
 $(\forall \text{inouts}_0. (\forall x. \text{length}(\text{inouts}_0 \ x) = (m-1)) (* \text{ For any } (m-1) \text{ inputs } *)$
 \longrightarrow
 $(\exists xx. (* \text{ there exists a signal } xx \text{ that is the } i1\text{th input and the } o1\text{th output } *)$
 $(\forall n. (xx \ n = (* \text{ the } o1\text{th output } *)$
 $(f (\lambda n1. f\text{-PreFD } xx \ i1 \ \text{inouts}_0 \ n1$
 $(* ((\text{take } i1 (\text{inouts}_0 \ n1)) \bullet (xx \ n1) \# (\text{drop } i1 (\text{inouts}_0 \ n1)))) *)$
 $(* \text{ assemble of inputs to make } xx \text{ as } i1\text{th } *)$
 $) \ n)!o1$
 $)$
 $)))))$

Solvable-unique: the feedback is solvable and has a unique solution.

definition *Solvable-unique*:: $\text{nat} (* \text{ the input index for feedback } *)$

$\Rightarrow \text{nat} (* \text{ the output index for feedback } *)$
 $\Rightarrow \text{nat} (* \text{ how many input signals } *)$
 $\Rightarrow \text{nat} (* \text{ how many output signals } *)$
 $\Rightarrow ((\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow \text{real list}) (* \text{ function } *)$
 $\Rightarrow \text{bool}$ **where**

$\text{Solvable-unique } i1 \ o1 \ m \ nn \ f = ((i1 < m \wedge o1 < nn) \wedge$
 $(\forall \text{inouts}_0. (\forall x. \text{length}(\text{inouts}_0 \ x) = (m-1)) (* \text{ For any } (m-1) \text{ inputs } *)$
 \longrightarrow
 $(\exists! (xx::\text{nat} \Rightarrow \text{real}). (* \text{ there only exists a signal } xx \text{ that is the } i1\text{th input and the } o1\text{th output } *)$
 $(\forall n. (xx \ n = (* \text{ the } o1\text{th output } *) (f (\lambda n1. f\text{-PreFD } xx \ i1 \ \text{inouts}_0 \ n1) \ n)!o1$
 $)$
 $)$
 $)$
 $)$

Solution returns the solution for a feedback block. Here the solution means the signal that could satisfy the feedback constraint (the related input is equal to the output)

definition *Solution*:: $\text{nat} (* \text{ the input index for feedback } *)$

$\Rightarrow \text{nat} (* \text{ the output index for feedback } *)$
 $\Rightarrow \text{nat} (* \text{ how many input signals } *)$
 $\Rightarrow \text{nat} (* \text{ how many output signals } *)$
 $\Rightarrow ((\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow \text{real list}) (* \text{ function } *)$
 $\Rightarrow (\text{nat} \Rightarrow \text{real list})$
 $\Rightarrow (\text{nat} \Rightarrow \text{real})$ **where**

$\text{Solution } i1 \ o1 \ m \ nn \ f \ \text{inouts}_0 =$
 $(\text{SOME } (xx::\text{nat} \Rightarrow \text{real}).$
 $((\forall x. \text{length}(\text{inouts}_0 \ x) = (m-1)) (* \text{ For any } (m-1) \text{ inputs } *)$

$$\begin{aligned} & \rightarrow *) \\ & (\forall n. (xx\ n = \\ & \quad (f\ (\lambda n1. f\text{-PreFD}\ xx\ i1\ inouts_0\ n1 \\ & \quad \quad (*\ ((take\ i1\ (inouts_0\ n1))\bullet[xx\ n1]\bullet(drop\ i1\ (inouts_0\ n1))))*) \\ & \quad \quad)\ n)!o1 \\ & \quad)) \\ &))) \end{aligned}$$

is-Solution checks if the supplied solution for a feedback block is a real solution.

definition *is-Solution*:: *nat* (* the input index for feedback *)

\Rightarrow *nat* (* the output index for feedback *)

\Rightarrow *nat* (* how many input signals *)

\Rightarrow *nat* (* how many output signals *)

\Rightarrow ((*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow *real list*) (* function *)

\Rightarrow ((*nat* \Rightarrow *real list*) \Rightarrow (*nat* \Rightarrow *real*))

\Rightarrow *bool* **where**

is-Solution *i1* *o1* *m* *nn* *f* *xx* = (

($\forall inouts_0. (\forall x. length(inouts_0\ x) = (m-1))$)

$\rightarrow (\forall n. (xx\ inouts_0\ n = (f\ (\lambda n1. f\text{-PreFD}\ (xx\ inouts_0)\ i1\ inouts_0\ n1)\ n)!o1))))$)

A.5.5 Split

definition *f-Split2*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**

[*f-blocks*]: *f-Split2* *x* *n* = [*hd*(*x* *n*), *hd*(*x* *n*)]

definition *Split2* :: *sim-state hrel-des* **where**

[*f-sim-blocks*]: *Split2* = *FBlock* ($\lambda x\ n. True$) 1 2 (*f-Split2*)

A.6 Blocks

A.6.1 Source

A.6.1.1 Constant Constant Block: no inputs and only one output.

definition *f-Const*:: *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**

[*f-blocks*]: *f-Const* *x0* *x* *n* = [*x0*]

definition *Const* :: *real* \Rightarrow *sim-state hrel-des* **where**

[*f-sim-blocks*]: *Const* *c0* = *FBlock* ($\lambda x\ n. True$) 0 1 (*f-Const* *c0*)

A.6.2 Unit Delay

Unit Delay block: one parameter (initial output), one input and one output. And the output is equal to previous input if it is not the initial output; otherwise it is equal to the initial output.

definition *f-UnitDelay*:: *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**

[*f-blocks*]: *f-UnitDelay* *x0* *x* *n* = [*if* *n* = 0 *then* *x0* *else* *hd*(*x* (*n*-1))]

definition *UnitDelay* :: *real* \Rightarrow *sim-state hrel-des* **where**

[*f-sim-blocks*]: *UnitDelay* *x0* = *FBlock* ($\lambda x\ n. True$) 1 1 (*f-UnitDelay* *x0*)

A.6.3 Discrete-Time Integrator

The Discrete-Time Integrator block: performs discrete-time integration or accumulation of signal. Integration (T=Ts) or Accumulation (T=1) methods: forward Euler, backward Euler, and trapezoidal methods.

DT-int-fw: integration by Forward Euler

fun *sum-by-fw-euler* :: *nat* \Rightarrow *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *real* **where**
sum-by-fw-euler 0 *x0* *K* *T* *x* = *x0* |
sum-by-fw-euler (*Suc* *m*) *x0* *K* *T* *x* =
(*sum-by-fw-euler* *m* *x0* *K* *T* *x*) + (*K* * *T* * (*hd*(*x m*)))

definition *f-DT-int-fw* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DT-int-fw* *x0* *K* *T* *x* *n* = [*sum-by-fw-euler* *n* *x0* *K* *T* *x*]

definition *DT-int-fw* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow
sim-state hrel-des **where**

[*f-sim-blocks*]: *DT-int-fw* *x0* *K* *T* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DT-int-fw* *x0* *K* *T*)

DT-int-bw: integration by Backward Euler (Initial condition setting is set to State)

fun *sum-by-bw-euler* :: *nat* \Rightarrow *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *real* **where**
sum-by-bw-euler 0 *x0* *K* *T* *x* = *x0* + (*K* * *T* * (*hd*(*x 0*))) |
sum-by-bw-euler (*Suc* *m*) *x0* *K* *T* *x* =
(*sum-by-bw-euler* *m* *x0* *K* *T* *x*) + (*K* * *T* * (*hd*(*x m*)))

definition *f-DT-int-bw* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DT-int-bw* *x0* *K* *T* *x* *n* = [*sum-by-bw-euler* *n* *x0* *K* *T* *x*]

definition *DT-int-bw* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow *sim-state hrel-des* **where**

[*f-sim-blocks*]: *DT-int-bw* *x0* *K* *T* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DT-int-bw* *x0* *K* *T*)

DT-int-trape: integration by Trapezoidal (Initial condition setting is set to State).

fun *sum-by-trape* **where**

sum-by-trape 0 *x0* *K* *T* *x* = *x0* + (*K* * (*T div* 2) * (*hd*(*x 0*))) |
sum-by-trape (*Suc* *m*) *x0* *K* *T* *x* =
(*sum-by-trape* *m* *x0* *K* *T* *x*) +
(*K* * (*T div* 2) * (*hd*(*x m*))) +
(*K* * (*T div* 2) * (*hd*(*x (Suc m)*)))

definition *f-DT-int-trape* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DT-int-trape* *x0* *K* *T* *x* *n* = [*sum-by-trape* *n* *x0* *K* *T* *x*]

definition *DT-int-trape* :: *real* \Rightarrow *real* \Rightarrow *real* \Rightarrow

sim-state hrel-des **where**

[*f-sim-blocks*]: *DT-int-trape* *x0* *K* *T* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DT-int-trape* *x0* *K* *T*)

A.6.4 Sum

The Sum block performs addition or subtraction on its inputs.

sum-by-sign: Summation or subtraction of a list according to their corresponding signs. It requires the length of inputs are equal to that of signs (true for +)

fun *sum-by-sign* **where**

sum-by-sign [] - = 0 |
sum-by-sign (*x#xs*) (*s#ss*) = (*if* *s* *then* (*sum-by-sign* *xs* *ss* + *x*) *else* (*sum-by-sign* *xs* *ss* - *x*))

definition *f-SumSub*:: *bool list* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**

[*f-blocks*]: *f-SumSub* *signs* *x* *n* = [*sum-by-sign* (*x n*) *signs*]

SumSub: summation or subtraction according to supplied signs.

definition *SumSub* :: *nat* \Rightarrow *bool list* \Rightarrow *sim-state hrel-des* **where**
 [*f-sim-blocks*]: *SumSub nr signs* = *FBlock* ($\lambda x n. \text{True}$) *nr 1* (*f-SumSub signs*)

Sum2 is a special case of *SumSub* and it adds up two inputs

definition *f-Sum2*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
 [*f-blocks*]: *f-Sum2 x n* = [*hd*(*x n*) + *hd*(*tl*(*x n*))]

definition *Sum2* :: *sim-state hrel-des* **where**
 [*f-sim-blocks*]: *Sum2* = *FBlock* ($\lambda x n. \text{True}$) *2 1* (*f-Sum2*)

SumSub2 is a special case of *SumSub* and it is equal to subtract the second input from the first input.

definition *f-SumSub2* :: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
 [*f-blocks*]: *f-SumSub2 x n* = [*hd*(*x n*) - *hd*(*tl*(*x n*))]

definition *SumSub2* :: *sim-state hrel-des* **where**
 [*f-sim-blocks*]: *SumSub2* = *FBlock* ($\lambda x n. \text{True}$) *2 1* (*f-SumSub2*)

SubSum2 is a special case of *SumSub* and it is equal to subtract the first input from the second input.

definition *f-SubSum2* :: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
 [*f-blocks*]: *f-SubSum2 x n* = [- *hd*(*x n*) + *hd*(*tl*(*x n*))]

definition *SubSum2* :: *sim-state hrel-des* **where**
 [*f-sim-blocks*]: *SubSum2* = *FBlock* ($\lambda x n. \text{True}$) *2 1* (*f-SubSum2*)

A.6.5 Product

The Product block performs multiplication and division.

not-divide-by-zero is a predicate in assumption. For signs, true denotes * and false for /.

fun *not-divide-by-zero* **where**
not-divide-by-zero [] = *True* |
not-divide-by-zero (*x#xs*) (*s#ss*) =
 (*HOL.conj* (*not-divide-by-zero xs ss*) (*if s then True else (x \neq 0)*))

product-by-sign: multiplies or divides by signs.

fun *product-by-sign* **where**
product-by-sign [] = *1* |
product-by-sign (*x#xs*) (*s#ss*) =
 (*if s then (product-by-sign xs ss * x) else (product-by-sign xs ss / x)*)

definition *f-ProdDiv* :: *bool list* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
 [*f-blocks*]: *f-ProdDiv signs x n* = [*product-by-sign* (*x n*) *signs*]

definition *f-no-div-by-zero* :: *bool list* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow *bool* **where**
 [*f-blocks*]: *f-no-div-by-zero signs x n* = *not-divide-by-zero* (*x n*) *signs*

ProdDiv has additional precondition that assumes all values of the divisor inputs are not equal to zero.

definition *ProdDiv* :: *nat* \Rightarrow *bool list* \Rightarrow *sim-state hrel-des* **where**
 [*f-sim-blocks*]: *ProdDiv nr signs* = *FBlock* ($\lambda x n. (\text{f-no-div-by-zero signs } x n)$) *nr 1* (*f-ProdDiv signs*)

Mul2 is a special case of *ProdDiv* and it multiplies two inputs.

definition *f-Mul2*:: (nat ⇒ real list) ⇒ nat ⇒ (real list) **where**
 [f-blocks]: *f-Mul2* *x n* = [hd(*x n*) * hd(tl(*x n*))]

definition *Mul2* :: sim-state hrel-des **where**
 [f-sim-blocks]: *Mul2* = FBlock (λ*x n*. True) 2 1 (*f-Mul2*)

Div2 is a special case of *ProdDiv* and the first input is divided by the second input.

definition *f-Div2*:: (nat ⇒ real list) ⇒ nat ⇒ (real list) **where**
 [f-blocks]: *f-Div2* *x n* = [hd(*x n*) / hd(tl(*x n*))]

definition *Div2* :: sim-state hrel-des **where**
 [f-sim-blocks]: *Div2* = FBlock (λ*x n*. (hd(tl(*x n*)) ≠ 0)) 2 1 (*f-Div2*)

A.6.6 Gain

definition *f-Gain*:: real ⇒ (nat ⇒ real list) ⇒ nat ⇒ (real list) **where**
 [f-blocks]: *f-Gain* *k x n* = [*k* * hd(*x n*)]

definition *Gain* :: real ⇒ sim-state hrel-des **where**
 [f-sim-blocks]: *Gain* *k* = FBlock (λ*x n*. True) 1 1 (*f-Gain* *k*)

A.6.7 Saturation

definition *f-Limit*:: real ⇒ real ⇒ (nat ⇒ real list) ⇒ nat ⇒ (real list) **where**
 [f-blocks]: *f-Limit* *ymin ymax x n* =
 [if *ymin* > hd(*x n*) then *ymin* else
 (if *ymax* < hd(*x n*) then *ymax* else hd(*x n*))]

definition *Limit* :: real ⇒ real ⇒ sim-state hrel-des **where**
 [f-sim-blocks]: *Limit* *ymin ymax* = FBlock (λ*x n*. True) 1 1 (*f-Limit* *ymin ymax*)

A.6.8 MinMax

MinList: return the minimum number from a list of numbers.

fun *MinList* **where**
MinList [] *minx* = *minx* |
MinList (*x#xs*) *minx* =
 (if *x* < *minx*
 then *MinList* *xs* *x*
 else *MinList* *xs* *minx*)

The input list must not be empty.

abbreviation *MinLst* ≡ (λ *lst* . *MinList* *lst* (hd(*lst*)))

MaxList: return the maximum number from a list of numbers.

fun *MaxList* **where**
MaxList [] *maxx* = *maxx* |
MaxList (*x#xs*) *maxx* =
 (if *x* > *maxx*
 then *MaxList* *xs* *x*
 else *MaxList* *xs* *maxx*)

The input list must not be empty.

abbreviation *MaxLst* ≡ (λ *lst* . *MaxList* *lst* (hd(*lst*)))

MinN returns the minimum value in the inputs.

definition *f-MinN*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-MinN* *x n* = [*MinLst* (*x n*)]

definition *MinN* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *MinN nr* = *FBlock* ($\lambda x n. True$) *nr 1* (*f-MinN*)

definition *f-Min2*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-Min2* *x n* = [*min* (*hd*(*x n*)) (*hd*(*tl*(*x n*)))]

definition *Min2* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *Min2* = *FBlock* ($\lambda x n. True$) *2 1* (*f-Min2*)

MaxN returns the maximum value in the inputs.

definition *f-MaxN*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-MaxN* *x n* = [*MaxLst* (*x n*)]

definition *MaxN* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *MaxN nr* = *FBlock* ($\lambda x n. True$) *nr 1* (*f-MaxN*)

definition *f-Max2*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-Max2* *x n* = [*max* (*hd*(*x n*)) (*hd*(*tl*(*x n*)))]

definition *Max2* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *Max2* = *FBlock* ($\lambda x n. True$) *2 1* (*f-Max2*)

A.6.9 Rounding

The Rounding Function block applies a rounding function to the input signal to produce the output signal.

RoundFloor rounds inputs using the floor function.

definition *f-RoundFloor*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-RoundFloor* *x n* = [*real-of-int* [(*hd*(*x n*))]]

definition *RoundFloor* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *RoundFloor* = *FBlock* ($\lambda x n. True$) *1 1* (*f-RoundFloor*)

RoundCeil rounds inputs using the ceil function.

definition *f-RoundCeil*:: $(nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**
[f-blocks]: *f-RoundCeil* *x n* = [*real-of-int* [(*hd*(*x n*))]]

definition *RoundCeil* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *RoundCeil* = *FBlock* ($\lambda x n. True$) *1 1* (*f-RoundCeil*)

A.6.10 Logic Operators

The Logical Operator block performs the specified logical operation on its inputs.

- It supports seven operators: AND, OR, NAND, NOR, XOR, NXOR, NOT;
- An input value is TRUE (1) if it is nonzero and FALSE (0) if it is zero;
- An output value is 1 if TRUE and 0 if FALSE;

A.6.10.1 AND **fun** $LAnd :: real\ list \Rightarrow bool$ **where**

$LAnd [] = True$ |

$LAnd (x\#\#xs) = (if\ x = 0\ then\ False\ else\ (LAnd\ xs))$

definition $f-LopAND :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f-blocks]: f-LopAND\ x\ n = [if\ LAnd\ (x\ n)\ then\ 1\ else\ 0]$

definition $LopAND :: nat \Rightarrow sim-state\ hrel-des$ **where**

$[f-sim-blocks]: LopAND\ m = FBlock\ (\lambda x\ n.\ True)\ m\ 1\ (f-LopAND)$

A.6.10.2 OR **fun** $LOr :: real\ list \Rightarrow bool$ **where**

$LOr [] = False$ |

$LOr (x\#\#xs) = (if\ x \neq 0\ then\ True\ else\ (LOr\ xs))$

definition $f-LopOR :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f-blocks]: f-LopOR\ x\ n = [if\ LOr\ (x\ n)\ then\ 1\ else\ 0]$

definition $LopOR :: nat \Rightarrow sim-state\ hrel-des$ **where**

$[f-sim-blocks]: LopOR\ m = FBlock\ (\lambda x\ n.\ True)\ m\ 1\ (f-LopOR)$

A.6.10.3 NAND **fun** $LNand :: real\ list \Rightarrow bool$ **where**

$LNand [] = False$ |

$LNand (x\#\#xs) = (if\ x = 0\ then\ True\ else\ (LNand\ xs))$

definition $f-LopNAND :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f-blocks]: f-LopNAND\ x\ n = [if\ LNand\ (x\ n)\ then\ 1\ else\ 0]$

definition $LopNAND :: nat \Rightarrow sim-state\ hrel-des$ **where**

$[f-sim-blocks]: LopNAND\ m = FBlock\ (\lambda x\ n.\ True)\ m\ 1\ (f-LopNAND)$

A.6.10.4 NOR **fun** $LNor :: real\ list \Rightarrow bool$ **where**

$LNor [] = True$ |

$LNor (x\#\#xs) = (if\ x \neq 0\ then\ False\ else\ (LNand\ xs))$

definition $f-LopNOR :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f-blocks]: f-LopNOR\ x\ n = [if\ LNor\ (x\ n)\ then\ 1\ else\ 0]$

definition $LopNOR :: nat \Rightarrow sim-state\ hrel-des$ **where**

$[f-sim-blocks]: LopNOR\ m = FBlock\ (\lambda x\ n.\ True)\ m\ 1\ (f-LopNOR)$

A.6.10.5 XOR **fun** $LXor :: real\ list \Rightarrow nat \Rightarrow bool$ **where**

$LXor []\ t = (if\ t\ mod\ 2 = 0\ then\ False\ else\ True)$ |

$LXor (x\#\#xs)\ t = (if\ x \neq 0\ then\ (LXor\ xs\ (t+1))\ else\ (LXor\ xs\ t))$

lemma $LXor\ [0,\ 1,\ 1]\ 0 = False$

by *auto*

lemma $LXor\ [0,\ 1,\ 1,\ 1]\ 0 = True$

by *auto*

definition $f-LopXOR :: (nat \Rightarrow real\ list) \Rightarrow nat \Rightarrow (real\ list)$ **where**

$[f-blocks]: f-LopXOR\ x\ n = [if\ LXor\ (x\ n)\ 0\ then\ 1\ else\ 0]$

definition $LopXOR :: nat \Rightarrow sim-state\ hrel-des$ **where**

$[f-sim-blocks]: LopXOR\ m = FBlock\ (\lambda x\ n.\ True)\ m\ 1\ (f-LopXOR)$

A.6.10.6 NXOR **fun** $LNxor :: \text{real list} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**
 $LNxor [] t = (\text{if } t \bmod 2 = 0 \text{ then True else False}) |$
 $LNxor (x\#\#xs) t = (\text{if } x \neq 0 \text{ then } (LNxor xs (t+1)) \text{ else } (LNxor xs t))$

lemma $LNxor [0, 1, 1] 0 = \text{True}$
by *auto*

lemma $LNxor [0, 1, 1, 1] 0 = \text{False}$
by *auto*

definition $f\text{-LopNXOR} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-LopNXOR } x n = [\text{if } LNxor (x n) 0 \text{ then } 1 \text{ else } 0]$

definition $LopNXOR :: \text{nat} \Rightarrow \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: LopNXOR m = FBlock (\lambda x n. \text{True}) m 1 (f\text{-LopNXOR})$

A.6.10.7 NOT **definition** $f\text{-LopNOT} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-LopNOT } x n = [\text{if } hd(x n) = 0 \text{ then } 1 \text{ else } 0]$

definition $LopNOT :: \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: LopNOT = FBlock (\lambda x n. \text{True}) 1 1 (f\text{-LopNOT})$

A.6.11 Relational Operator

The Relational Operator block performs specified relational operation on inputs.

- It supports six operators for two-input mode: $==$, $=$, $<$, $<=$, $>$, $>=$;
- An output value is 1 if TRUE and 0 if FALSE;

A.6.11.1 Equal $==$ **definition** $f\text{-RopEQ} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-RopEQ } x n = [\text{if } hd(x n) = hd(tl(x n)) \text{ then } 1 \text{ else } 0]$

definition $RopEQ :: \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: RopEQ = FBlock (\lambda x n. \text{True}) 2 1 (f\text{-RopEQ})$

A.6.11.2 Notequal $=$ **definition** $f\text{-RopNEQ} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-RopNEQ } x n = [\text{if } hd(x n) = hd(tl(x n)) \text{ then } 0 \text{ else } 1]$

definition $RopNEQ :: \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: RopNEQ = FBlock (\lambda x n. \text{True}) 2 1 (f\text{-RopNEQ})$

A.6.11.3 Less Than $<$ **definition** $f\text{-RopLT} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-RopLT } x n = [\text{if } hd(x n) < hd(tl(x n)) \text{ then } 1 \text{ else } 0]$

definition $RopLT :: \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: RopLT = FBlock (\lambda x n. \text{True}) 2 1 (f\text{-RopLT})$

A.6.11.4 Less Than or Equal to $<=$ **definition** $f\text{-RopLE} :: (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]: f\text{-RopLE } x n = [\text{if } hd(x n) \leq hd(tl(x n)) \text{ then } 1 \text{ else } 0]$

definition $RopLE :: \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]: RopLE = FBlock (\lambda x n. \text{True}) 2 1 (f\text{-RopLE})$

A.6.11.5 Greater Than > definition $f\text{-RopGT}$: $(\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]$: $f\text{-RopGT } x \ n = [\text{if } \text{hd}(x \ n) > \text{hd}(\text{tl}(x \ n)) \text{ then } 1 \text{ else } 0]$

definition RopGT : $\text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{RopGT} = \text{FBlock } (\lambda x \ n. \text{True}) \ 2 \ 1 \ (f\text{-RopGT})$

A.6.11.6 Greater Than or Equal to \geq definition $f\text{-RopGE}$: $(\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]$: $f\text{-RopGE } x \ n = [\text{if } \text{hd}(x \ n) \geq \text{hd}(\text{tl}(x \ n)) \text{ then } 1 \text{ else } 0]$

definition RopGE : $\text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{RopGE} = \text{FBlock } (\lambda x \ n. \text{True}) \ 2 \ 1 \ (f\text{-RopGE})$

A.6.12 Switch

The Switch block switches the output between the first input and the third input based on the value of the second input.

- The first and the third inputs are data inputs;
- The second is the control input.
- Criteria for passing first input: $u2 \geq \text{Threshold}$, $u2 > \text{Threshold}$, or $u2 = 0$;

Switch1 : criteria is $u2 \geq \text{Threshold}$

definition $f\text{-Switch1}$: $\text{real} \Rightarrow (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]$: $f\text{-Switch1 } th \ x \ n = [\text{if } (x \ n)!1 \geq th \text{ then } (x \ n)!0 \text{ else } (x \ n)!2]$

definition Switch1 : $\text{real} \Rightarrow \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{Switch1 } th = \text{FBlock } (\lambda x \ n. \text{True}) \ 3 \ 1 \ (f\text{-Switch1 } th)$

Switch2 : criteria is $u2 > \text{Threshold}$

definition $f\text{-Switch2}$: $\text{real} \Rightarrow (\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]$: $f\text{-Switch2 } th \ x \ n = [\text{if } (x \ n)!1 > th \text{ then } (x \ n)!0 \text{ else } (x \ n)!2]$

definition Switch2 : $\text{real} \Rightarrow \text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{Switch2 } th = \text{FBlock } (\lambda x \ n. \text{True}) \ 3 \ 1 \ (f\text{-Switch2 } th)$

Switch3 : criteria is $u2 = 0$

definition $f\text{-Switch3}$: $(\text{nat} \Rightarrow \text{real list}) \Rightarrow \text{nat} \Rightarrow (\text{real list})$ **where**
 $[f\text{-blocks}]$: $f\text{-Switch3 } x \ n = [\text{if } (x \ n)!1 = 0 \text{ then } (x \ n)!2 \text{ else } (x \ n)!0]$

definition Switch3 : $\text{sim-state hrel-des}$ **where**
 $[f\text{-sim-blocks}]$: $\text{Switch3} = \text{FBlock } (\lambda x \ n. \text{True}) \ 3 \ 1 \ (f\text{-Switch3})$

A.6.13 Data Type Conversion

Data Type Conversion: converts an input signal to the specified data type.

Integer round number towards zero

definition RoundZero : $\text{real} \Rightarrow \text{int}$ **where**
 $\text{RoundZero } x = (\text{if } x \geq (0::\text{real}) \text{ then } \lfloor x \rfloor \text{ else } \lceil x \rceil)$

lemma *RoundZero* $1.1 = 1$
apply (*simp add: RoundZero-def*)
done

lemma *RoundZero* $(-1.1) = -1$
apply (*simp add: RoundZero-def*)
done

int8: convert int to int8.

definition *int8* :: *int* \Rightarrow *int* **where**
int8 $x = ((x+128) \bmod 256) - 128$

int16: convert int to int16.

definition *int16* :: *int* \Rightarrow *int* **where**
int16 $x = ((x+32768) \bmod 65536) - 32768$

int32: convert int to int32.

definition *int32* :: *int* \Rightarrow *int* **where**
int32 $x = ((x+2147483648) \bmod 4294967296) - 2147483648$

lemma *int32-eq*:
assumes $x \geq 0 \wedge x < 2147483648$
shows *int32* $x = x$
apply (*simp add: int32-def*)
using *assms* **by** (*smt int-mod-eq*)

lemma *int8* $(-1) = -1$
by (*simp add: int8-def*)

lemma *int8* $(-128) = -128$
by (*simp add: int8-def*)

lemma *int8* $(-129) = 127$
by (*simp add: int8-def*)

lemma *int8* $(129) = -127$
by (*simp add: int8-def*)

lemma *int8* $(-378) = -122$
by (*simp add: int8-def*)

lemma *int8* $(378) = 122$
by (*simp add: int8-def*)

uint8: convert int to uint8

definition *uint8* :: *int* \Rightarrow *int* **where**
uint8 $x = x \bmod 256$

lemma *uint8* $(-1) = 255$
by (*simp add: uint8-def*)

uint16: convert int to uint16

definition *uint16* :: *int* \Rightarrow *int* **where**
uint16 $x = x \bmod 65536$

uint32: convert int to uint32

definition *uint32* :: *int* \Rightarrow *int* **where**
uint32 *x* = *x* mod 4294967296

lemma (*uint32* 4294967296) = 0
by (*simp add: uint32-def*)

lemma (*uint32* 4294967295) = 4294967295
by (*simp add: uint32-def*)

lemma (*uint32* (-1)) = 4294967295
by (*simp add: uint32-def*)

lemma (*uint32* (-4294967298)) = 4294967294
by (*simp add: uint32-def*)

DataTypeConvUint32Zero: convert to uint32 and round number towards zero.

definition *f-DTConvUint32Zero*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvUint32Zero* *x n* = [real-of-int (*uint32* (*RoundZero*(*hd* (*x n*))))]

definition *DataTypeConvUint32Zero* :: *sim-state hrel-des* **where**
[*f-sim-blocks*]: *DataTypeConvUint32Zero* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvUint32Zero*)

DataTypeConvInt32Zero: convert to int32 and round number towards zero.

definition *f-DTConvInt32Zero*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvInt32Zero* *x n* = [real-of-int (*int32* (*RoundZero*(*hd* (*x n*))))]

definition *DataTypeConvInt32Zero* :: *sim-state hrel-des* **where**
[*f-sim-blocks*]: *DataTypeConvInt32Zero* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvInt32Zero*)

DataTypeConvUint32Floor: convert to uint32 and round number using floor.

definition *f-DTConvUint32Floor*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvUint32Floor* *x n* = [real-of-int (*uint32* ($\lfloor \text{hd} (x n) \rfloor$)))]

definition *DataTypeConvUint32Floor* :: *sim-state hrel-des* **where**
[*f-sim-blocks*]: *DataTypeConvUint32Floor* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvUint32Floor*)

DataTypeConvInt32Floor: convert to int32 and round number using floor.

definition *f-DTConvInt32Floor*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvInt32Floor* *x n* = [real-of-int (*int32* ($\lfloor \text{hd} (x n) \rfloor$)))]

definition *DataTypeConvInt32Floor* :: *sim-state hrel-des* **where**
[*f-sim-blocks*]: *DataTypeConvInt32Floor* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvInt32Floor*)

DataTypeConvUint32Ceil: convert to uint32 and round number using ceil.

definition *f-DTConvUint32Ceil*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvUint32Ceil* *x n* = [real-of-int (*uint32* ($\lceil \text{hd} (x n) \rceil$)))]

definition *DataTypeConvUint32Ceil* :: *sim-state hrel-des* **where**
[*f-sim-blocks*]: *DataTypeConvUint32Ceil* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvUint32Ceil*)

DataTypeConvInt32Ceil: convert to int32 and round number using ceil.

definition *f-DTConvInt32Ceil*:: (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[*f-blocks*]: *f-DTConvInt32Ceil* *x n* = [real-of-int (*int32* ($\lceil \text{hd} (x n) \rceil$)))]

definition *DataTypeConvInt32Ceil* :: *sim-state hrel-des* **where**
[f-sim-blocks]: *DataTypeConvInt32Ceil* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-DTConvInt32Ceil*)

A.6.14 Initial Condition (IC)

The IC block sets the initial condition of the signal at its input port. The block does this by outputting the specified initial condition when you start the simulation, regardless of the actual value of the input signal. Thereafter, the block outputs the actual value of the input signal.

definition *f-IC* :: *real* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[f-blocks]: *f-IC* *x0* *x* *n* = [*if* *n* = 0 *then* *x0* *else* *hd*(*x* *n*)]

definition *IC* :: *real* \Rightarrow *sim-state hrel-des* **where**
[f-sim-blocks]: *IC* *x0* = *FBlock* ($\lambda x n. \text{True}$) 1 1 (*f-IC* *x0*)

A.6.15 Router Block

A new introduced block to route signals: the same number of inputs and outputs but in different orders.

fun *assembleOutput* :: *real list* \Rightarrow *nat list* \Rightarrow *real list* **where**
assembleOutput *ins* [] = [] |
assembleOutput *ins* (*x* # *xs*) = (*ins*! *x*) # (*assembleOutput* *ins* (*xs*))

definition *f-Router* :: *nat list* \Rightarrow (*nat* \Rightarrow *real list*) \Rightarrow *nat* \Rightarrow (*real list*) **where**
[f-blocks]: *f-Router* *routes* *x* *n* = *assembleOutput* (*x* *n*) *routes*

lemma *f-Router* [2,0,1] ($\lambda na. [11, 22, 33]$) *n* = [33, 11, 22]
by (*simp add: f-blocks*)

definition *Router* :: *nat* \Rightarrow *nat list* \Rightarrow *sim-state hrel-des* **where**
[f-sim-blocks]: *Router* *nn* *routes* = *FBlock* ($\lambda x n. \text{True}$) *nn* *nn* (*f-Router* *routes*)

end

B Block Laws

In this section, many theorems and laws are proved to facilitate application of our theories in Simulink block diagrams.

theory *simu-contract-real-laws*

imports

simu-contract-real

begin

— timeout in seconds

declare [[*smt-timeout* = 600]]

B.1 Additional Laws

list-len-avail: there always exists some signals that could have a specific size.

lemma *list-len-avail*:

$\forall x \geq 0. (\exists (xx :: \text{nat} \Rightarrow \text{real list}). \forall n. \text{length } (xx \ n) = x)$

```

apply (rule allI)
apply (auto)
apply (induct-tac x)
apply (rule-tac x = λna. [] in exI, simp)
apply (auto)
by (rule-tac x = λna. 0#(xx na) in exI, simp)

```

list-len-avail: there always exists some signals that could have a specific size and the value of each signal is equal to an arbitrary real number.

lemma *list-len-avail'*:

```

∀ r::real. ∀ x≥0. (∃ (xx::nat⇒real list). (∀ n. (length (xx n) = x) ∧ (∀ y::nat<x. ((xx n)!y = r))))
apply (rule allI)
apply (auto)
apply (induct-tac x)
apply (rule-tac x = λna. [] in exI, simp)
apply (auto)
apply (rule-tac x = λna. r#(xx na) in exI, simp)
using less-Suc-eq-0-disj by auto

```

sum-hd-signal sums up a signal's current value and all past values.

```

fun sum-hd-signal:: (nat ⇒ real list) ⇒ nat ⇒ real where
sum-hd-signal x 0 = hd(x 0) |
sum-hd-signal x (Suc n) = hd(x (Suc n)) + sum-hd-signal x (n)

```

remove-at removes the *i*th element from a list.

abbreviation *remove-at* ≡ (λlst i. (take (i) lst)•(drop (i+1) lst))

lemma *remove-at* [] 1 = [] **by** simp

lemma *remove-at* [2,3,4] 1 = [2,4] **by** simp

fun-eq: two functions are equal as long as they are equal in all their domains (total functions).

lemma *fun-eq*:

```

assumes ∀ x. f x = g x
shows f = g
by (simp add: assms ext)

```

fun-eq': two functions are equal in all their domains then they are equal functions. (total functions).

lemma *fun-eq'*:

```

assumes f = g
shows ∀ x. (f x = g x)
by (simp add: assms)

```

lemma *fun-neq*:

```

assumes ∀ x. ¬ (f x = g x)
shows ¬ f = g
using assms by auto

```

ref-eq: two predicates are equal as long as they are refined by each other.

lemma *ref-eq*:

```

assumes P ⊆ Q
assumes Q ⊆ P
shows P = Q

```


by (simp add: antisym assms(1) assms(2))

lemma *hd-drop-m*:

$\forall (x::nat \Rightarrow \text{real list})\ n::nat.\ \text{length}(x\ n) > m \longrightarrow (\text{hd}(\text{drop}\ m\ (x\ n)) = x\ n!m)$
 using *hd-drop-conv-nth* by *blast*

lemma *hd-take-m*:

$m > 0 \longrightarrow (\forall (x::nat \Rightarrow \text{real list})\ n::nat.\ (\text{hd}(\text{take}\ m\ (x\ n)) = \text{hd}(x\ n)))$
 by (metis *append-take-drop-id* *hd-append2* *less-numeral-extra(3)* *take-eq-Nil*)

lemma *hd-tl-take-m*:

$m > 1 \longrightarrow (\forall (x::nat \Rightarrow \text{real list})\ n::nat.\ (\text{hd}(\text{tl}(\text{take}\ m\ (x\ n)))) = \text{hd}(\text{tl}(x\ n)))$
 by (metis *hd-conv-nth* *less-numeral-extra(3)* *nth-take* *take-eq-Nil* *tl-take* *zero-less-diff*)

B.2 SimBlock healthiness

lemma *SimBlock-FBlock* [*simblock-healthy*]:

assumes *s1*: $\exists \text{inouts}_v\ \text{inouts}'_v$.
 $\forall x.\ \text{length}(\text{inouts}'_v\ x) = n \wedge$
 $\text{length}(\text{inouts}_v\ x) = m \wedge$
 $f\ \text{inouts}_v\ x = \text{inouts}'_v\ x$
assumes *s2*: $\forall x\ na.\ \text{length}(x\ na) = m \longrightarrow \text{length}(f\ x\ na) = n$
shows *SimBlock* *m* *n* (*FBlock* $(\lambda x\ n.\ \text{True})\ m\ n\ f$)
apply (simp add: *SimBlock-def* *FBlock-def*)
apply (rel-auto)
using *s1* **apply** *blast*
by (simp add: *s2*)

lemma *SimBlock-FBlock'* [*simblock-healthy*]:

assumes *s1*: $\exists \text{inouts}_v.\ (\forall x.\ p1\ \text{inouts}_v\ x) \wedge$
 $(\exists \text{inouts}'_v$.
 $\forall x.\ \text{length}(\text{inouts}'_v\ x) = n \wedge$
 $\text{length}(\text{inouts}_v\ x) = m \wedge$
 $f\ \text{inouts}_v\ x = \text{inouts}'_v\ x)$
assumes *s2*: $\forall x\ na.\ \text{length}(x\ na) = m \longrightarrow \text{length}(f\ x\ na) = n$
shows *SimBlock* *m* *n* (*FBlock* $(p1)\ m\ n\ f$)
apply (simp add: *SimBlock-def* *FBlock-def*)
apply (rel-auto)
using *s1* *s2* **by** *blast*

lemma *SimBlock-FBlock-fn* [*simblock-healthy*]:

assumes *s1*: *SimBlock* *m* *n* (*FBlock* $(\lambda x\ n.\ \text{True})\ m\ n\ f$)
shows $(\forall x\ xa.\ \text{length}(x\ xa) = m \longrightarrow \text{length}(f\ x\ xa) = n)$
proof –
have *1*: $\text{PrePost}((\text{FBlock}\ (\lambda x\ n.\ \text{True})\ m\ n\ f)) \neq \text{false}$
using *s1* *SimBlock-def*
by *blast*
then show *?thesis*
apply (simp add: *FBlock-def*)
apply (rel-simp)
done
qed

lemma *SimBlock-FBlock-fn'* [*simblock-healthy*]:

assumes *s1*: *SimBlock* *m* *n* (*FBlock* $(p)\ m\ n\ f$)
shows $(\forall x\ xa.\ \text{length}(x\ xa) = m \longrightarrow \text{length}(f\ x\ xa) = n)$

proof –
 have 1: $PrePost((FBlock\ (p)\ m\ n\ f)) \neq false$
 using s1 *SimBlock-def*
 by *blast*
 then show ?thesis
 apply (simp add: *FBlock-def*)
 apply (*rel-simp*)
 done
qed

lemma *SimBlock-FBlock-p* [*simblock-healthy*]:
 assumes s1: *SimBlock* m n (*FBlock* (p) m n f)
 shows $\exists inouts_v. \forall x. p\ inouts_v\ x \wedge length(inouts_v\ x) = m$
proof –
 have 1: $PrePost((FBlock\ (p)\ m\ n\ f)) \neq false$
 using s1 *SimBlock-def*
 by *blast*
 then show ?thesis
 apply (simp add: *FBlock-def*)
 apply (*rel-simp*)
 by *blast*
qed

lemma *SimBlock-FBlock-p-f* [*simblock-healthy*]:
 assumes s1: *SimBlock* m n (*FBlock* (p) m n f)
 shows $\exists inouts_v. \forall x. p\ inouts_v\ x \wedge$
 $(\exists inouts_v'. \forall x. length(inouts_v'\ x) = n \wedge length(inouts_v\ x) = m \wedge f\ inouts_v\ x = inouts_v'\ x)$
proof –
 have 1: $PrePost((FBlock\ (p)\ m\ n\ f)) \neq false$
 using s1 *SimBlock-def*
 by *blast*
 then show ?thesis
 apply (simp add: *FBlock-def*)
 apply (*rel-simp*)
 by *blast*
qed

lemma *FBlock-eq*:
 assumes f1 = f2
 shows *FBlock* p-f m n f1 = *FBlock* p-f m n f2
 using *assms* by *simp*

lemma *FBlock-eq'*:
 assumes $\forall (x::nat \Rightarrow real\ list)\ n. length(x\ n) = m \longrightarrow f1\ x\ n = f2\ x\ n$
 shows *FBlock* p-f m n f1 = *FBlock* p-f m n f2
 apply (simp add: *FBlock-def*)
 apply (*rule* *ref-eq*)
 apply (*rel-simp*)
 using *assms* apply *simp*
 apply (*rel-simp*)
 using *assms* by *metis*

lemma *FBlock-eq''*:
 assumes s1: $\forall (x::nat \Rightarrow real\ list)\ n. (\forall na. length(x\ na) = m) \longrightarrow f1\ x\ n = f2\ x\ n$

```

assumes s2:  $\forall (x::nat \Rightarrow real\ list)\ na.\ length(f1\ x\ na) = n$ 
assumes s3:  $\forall (x::nat \Rightarrow real\ list)\ na.\ length(f2\ x\ na) = n$ 
shows  $FBlock\ p\ f\ m\ n\ f1 = FBlock\ p\ f\ m\ n\ f2$ 
apply (simp add: FBlock-def)
apply (rule ref-eq)
apply (rel-simp)
apply (rule conjI)
apply (simp add: assms)
using assms apply blast
apply (rel-simp)
using assms by metis

```

B.3 inps and outps

lemma *inps-P*:

```

assumes SimBlock m n P
shows  $inps\ P = m$ 
using assms inps-outps by auto

```

lemma *outps-P*:

```

assumes SimBlock m n P
shows  $outps\ P = n$ 
using assms inps-outps by auto

```

lemma *SimBlock-implies-not-PQ* [*simblock-healthy*]:

```

assumes  $s1: SimBlock\ m\ n\ (P \vdash_n Q)$ 
shows  $([P]_{<} \wedge Q) \neq false$ 
using SimBlock-def s1 by auto

```

lemma *SimBlock-implies-not-P* [*simblock-healthy*]:

```

assumes  $s1: SimBlock\ m\ n\ (P \vdash_n Q)$ 
shows  $[P]_{<} \neq false$ 
using SimBlock-def s1
by (metis SimBlock-implies-not-PQ aext-false ndesign-def ndesign-refinement' true-conj-zero(1)
utp-pred-laws.bot.extremum utp-pred-laws.inf.orderE)

```

lemma *SimBlock-implies-not-P'* [*simblock-healthy*]:

```

assumes  $s1: SimBlock\ m\ n\ (P \vdash_n Q)$ 
shows  $P \neq false$ 
using SimBlock-def s1
by (metis SimBlock-implies-not-PQ aext-false ndesign-def
utp-pred-laws.bot.extremum utp-pred-laws.inf.orderE)

```

lemma *SimBlock-implies-not-P''* [*simblock-healthy*]:

```

assumes  $s1: SimBlock\ m\ n\ (P \vdash_n Q)$ 
shows  $\exists inouts_v\ inouts_v'. \llbracket [P]_{<} \rrbracket_e ((inouts_v = inouts_v), (inouts_v = inouts_v'))$ 
using SimBlock-implies-not-P
by (metis (mono-tags, hide-lams) bot-bool-def bot-uepr.rep-eq false-upred-def old.unit.exhaust s1
sim-state.cases-scheme surj-pair udeduct-eqI)

```

lemma *SimBlock-implies-not-P-cond* [*simblock-healthy*]:

```

assumes  $s1: SimBlock\ m\ n\ (P \vdash_r Q)$ 
assumes  $s2: out\alpha \nmid P$ 
shows  $\forall inouts_v\ inouts_v'\ inouts_v''. \llbracket [P] \rrbracket_e ((inouts_v = inouts_v), (inouts_v = inouts_v')) = \llbracket [P] \rrbracket_e ((inouts_v = inouts_v), (inouts_v = inouts_v'))$ 

```

using *SimBlock-implies-not-P s1 s2*
by (*rel-simp*)

lemma *SimBlock-implies-not-Q [simblock-healthy]*:
assumes *s1: SimBlock m n (P ⊢_n Q)*
shows *Q ≠ false*
using *SimBlock-def s1* **by** *auto*

lemma *SimBlock-implies-not-Q' [simblock-healthy]*:
assumes *s1: SimBlock m n (P ⊢_n Q)*
shows $\exists \text{inouts}_v \text{inouts}'_v. \llbracket Q \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v))$
using *SimBlock-implies-not-Q*
by (*metis (mono-tags, hide-lams) bot-bool-def bot-uepr.rep-eq false-upred-def old.unit.exhaust s1 sim-state.cases-scheme surj-pair udeduct-eqI*)

lemma *SimBlock-implies-not-PQ' [simblock-healthy]*:
assumes *s1: SimBlock m n (P ⊢_n Q)*
shows $\exists \text{inouts}_v \text{inouts}'_v. (\llbracket P \rrbracket_e ((\text{inouts}_v = \text{inouts}_v)) \wedge \llbracket Q \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v)))$
using *s1 SimBlock-implies-not-PQ* **apply** (*rel-simp*)
done

lemma *SimBlock-implies-mP [simblock-healthy]*:
assumes *s1: SimBlock m n (P ⊢_n Q)*
shows $\forall \text{inouts}_v \text{inouts}'_v x. \llbracket P \rrbracket_e ((\text{inouts}_v = \text{inouts}_v)) \longrightarrow \llbracket Q \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v)) \longrightarrow \text{length}(\text{inouts}_v x) = m$
proof –
from *s1* **have** $1: ((\forall na \cdot \#_u(\&\text{inouts}(\langle\langle na \rangle\rangle)_a) =_u \langle\langle m \rangle\rangle) \sqsubseteq \text{Dom}(\text{PrePost}((P \vdash_n Q))))$
by (*simp add: SimBlock-def*)
then show *?thesis*
by (*rel-auto*)
qed

lemma *SimBlock-implies-Qn [simblock-healthy]*:
assumes *s1: SimBlock m n (P ⊢_n Q)*
shows $\forall \text{inouts}_v \text{inouts}'_v x. \llbracket P \rrbracket_e ((\text{inouts}_v = \text{inouts}_v)) \longrightarrow \llbracket Q \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v)) \longrightarrow \text{length}(\text{inouts}'_v x) = n$
proof –
from *s1* **have** $1: ((\forall na \cdot \#_u(\&\text{inouts}(\langle\langle na \rangle\rangle)_a) =_u \langle\langle n \rangle\rangle) \sqsubseteq \text{Ran}(\text{PrePost}((P \vdash_n Q))))$
by (*simp add: SimBlock-def*)
then show *?thesis*
by (*rel-auto*)
qed

lemma *sim-refine-implies-inps-outps-eq*:
assumes *s1: SimBlock m1 n1 (P)*
assumes *s2: SimBlock m2 n2 (Q)*
assumes *s3: (P) ⊆ (Q)*
assumes *s4: (pre_D(P) ∧ post_D(Q)) ≠ false*
shows *m1 = m2 ∧ n1 = n2*
proof –

```

have ref-des:  $pre_D(Q) \sqsubseteq pre_D(P) \wedge post_D(P) \sqsubseteq (pre_D(P) \wedge post_D(Q))$ 
  using s3
  by (simp add: design-refine-thms(1) design-refine-thms(2) refBy-order)
have pred-1:  $PrePost(P) = (pre_D(P) \wedge post_D(P))$ 
  apply (simp)
done
have pred-2:  $PrePost(Q) = (pre_D(Q) \wedge post_D(Q))$ 
  apply (simp)
done
have pred-1-not-false:  $(pre_D(P) \wedge post_D(P)) \neq false$ 
  using SimBlock-def s1 by force
have pred-2-not-false:  $(pre_D(Q) \wedge post_D(Q)) \neq false$ 
  using SimBlock-def s2 by force
have ref-inps-1:  $((\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle) \sqsubseteq Dom((pre_D(P) \wedge post_D(P))))$ 
  using s1 apply (simp add: SimBlock-def)
done
then have ref-inps-12:  $\dots \sqsubseteq Dom((pre_D(P) \wedge post_D(Q)))$ 
  apply (simp add: ref-des Dom-def)
  by (smt ref-des arestr.rep-eq conj-upred-def ex.rep-eq inf-bool-def inf-ueq.rep-eq upred-ref-iff)
have ref-inps-2:  $((\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \sqsubseteq Dom((pre_D(Q) \wedge post_D(Q))))$ 
  using s2 apply (simp add: SimBlock-def)
done
have ref-p2-p1:  $Dom((pre_D(Q) \wedge post_D(Q))) \sqsubseteq Dom((pre_D(P) \wedge post_D(Q)))$ 
  apply (simp add: Dom-def)
  by (smt ref-des aext-mono arestr-and order-refl utp-pred-laws.ex-mono utp-pred-laws.inf.absorb-iff2
  utp-pred-laws.inf-mono)
  from ref-p2-p1 and ref-inps-2 have ref-inps-2-p1:  $((\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \sqsubseteq Dom((pre_D(P) \wedge post_D(Q))))$ 
  by simp
  from ref-inps-2-p1 have P1-Q2-implies-m2:  $(\forall b. \llbracket Dom((pre_D(P) \wedge post_D(Q)) \rrbracket_e b \longrightarrow \llbracket (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \rrbracket_e b)$ 
  apply (simp add: upred-ref-iff)
  done
  from ref-inps-12 have P1-Q2-implies-m1:  $(\forall b. \llbracket Dom((pre_D(P) \wedge post_D(Q)) \rrbracket_e b \longrightarrow \llbracket (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle) \rrbracket_e b)$ 
  apply (simp add: upred-ref-iff)
  done
  from P1-Q2-implies-m1 and P1-Q2-implies-m2 have P1-Q2-implies-m2-m1:
     $\forall b. \llbracket Dom((pre_D(P) \wedge post_D(Q)) \rrbracket_e b \longrightarrow (\llbracket (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \rrbracket_e b \wedge \llbracket (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle) \rrbracket_e b)$ 
    by blast
  then have P1-Q2-implies-m2-m1-1:  $\forall b. \llbracket Dom((pre_D(P) \wedge post_D(Q)) \rrbracket_e b \longrightarrow (\llbracket (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \wedge (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle) \rrbracket_e b)$ 
  by (simp add: conj-implies2)
  have forall-comb:  $((\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \wedge (\forall na \cdot \#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle)) =$ 
     $(\forall na \cdot ((\#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \wedge (\#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle)))$ 
  apply (rel-auto)
  done
  from P1-Q2-implies-m2-m1-1 have P1-Q2-implies-m2-m1-2:
     $\forall b. \llbracket Dom((pre_D(P) \wedge post_D(Q)) \rrbracket_e b \longrightarrow (\llbracket (\forall na \cdot ((\#_u(\&inouts(\langle na \rangle)_a) =_u \langle m2 \rangle) \wedge (\#_u(\&inouts(\langle na \rangle)_a) =_u \langle m1 \rangle))) \rrbracket_e b)$ 
  by (simp add: forall-comb)
  have m1-m2-eq:  $m2 = m1$ 
  proof (rule ccontr)

```

```

    assume ss1: m2 ≠ m1
    have conj-false: (∀ na · ((#u(&inouts(«na»)a) =u «m2») ∧ (#u(&inouts(«na»)a) =u «m1»)))
= false
    using ss1 apply (rel-auto)
  done
  have imp-false: ∀ b. [[Dom((preD(P) ∧ postD(Q)))]e b → ([[false]]e b)
    using P1-Q2-implies-m2-m1-2
    apply (simp add: conj-false)
  done
  have dom-false: Dom((preD(P) ∧ postD(Q))) = false
  by (metis imp-false true-conj-zero(2) udeduct-refineI utp-pred-laws.inf.orderE utp-pred-laws.inf-commute)
  have P1-Q2-false: (preD(P) ∧ postD(Q)) = false
    by (metis assume-Dom assume-false dom-false segr-left-zero)
  show False
    using s4 apply (simp add: P1-Q2-false)
  done
qed

have ref-inps-1': ((∀ na · #u(&inouts(«na»)a) =u «n1») ⊆ Ran((preD(P) ∧ postD(P))))
  using s1 apply (simp add: SimBlock-def)
done
then have ref-inps-12': ... ⊆ Ran((preD(P) ∧ postD(Q)))
  apply (simp add: ref-des Ran-def)
  by (smt ref-des arestr.rep-eq conj-upred-def ex.rep-eq inf-bool-def inf-ueq.rep-eq upred-ref-iff)
have ref-inps-2': ((∀ na · #u(&inouts(«na»)a) =u «n2») ⊆ Ran((preD(Q) ∧ postD(Q))))
  using s2 apply (simp add: SimBlock-def)
done
have ref-p2-p1': Ran((preD(Q) ∧ postD(Q))) ⊆ Ran((preD(P) ∧ postD(Q)))
  apply (simp add: Ran-def)
  by (smt ref-des aext-mono arestr-and order-refl utp-pred-laws.ex-mono utp-pred-laws.inf.absorb-iff2
  utp-pred-laws.inf-mono)
  from ref-p2-p1' and ref-inps-2' have ref-inps-2-p1': ((∀ na · #u(&inouts(«na»)a) =u «n2») ⊆
Ran((preD(P) ∧ postD(Q))))
  by simp
  from ref-inps-2-p1' have P1-Q2-implies-n2: (∀ b. [[Ran((preD(P) ∧ postD(Q)))]e b → [[(∀ na ·
#u(&inouts(«na»)a) =u «n2»]]e b)
  apply (simp add: upred-ref-iff)
done
  from ref-inps-12' have P1-Q2-implies-n1: (∀ b. [[Ran((preD(P) ∧ postD(Q)))]e b → [[(∀ na ·
#u(&inouts(«na»)a) =u «n1»]]e b)
  apply (simp add: upred-ref-iff)
done
  from P1-Q2-implies-n1 and P1-Q2-implies-n2 have P1-Q2-implies-n2-n1:
  ∀ b. [[Ran((preD(P) ∧ postD(Q)))]e b → ([[(∀ na · #u(&inouts(«na»)a) =u «n2»]]e b ∧ [[(∀ na
· #u(&inouts(«na»)a) =u «n1»]]e b)
  by blast
  then have P1-Q2-implies-n2-n1-1:
  ∀ b. [[Ran((preD(P) ∧ postD(Q)))]e b → ([[(∀ na · #u(&inouts(«na»)a) =u «n2»)]e b ∧ (∀ na ·
#u(&inouts(«na»)a) =u «n1»)]e b)
  by (simp add: conj-implies2)
  have forall-comb': ((∀ na · #u(&inouts(«na»)a) =u «n2») ∧ (∀ na · #u(&inouts(«na»)a) =u
«n1»)) =
  (∀ na · ((#u(&inouts(«na»)a) =u «n2») ∧ (#u(&inouts(«na»)a) =u «n1»)))
  apply (rel-auto)
done

```

```

from P1-Q2-implies-n2-n1-1 have P1-Q2-implies-n2-n1-2:
   $\forall b. \llbracket \text{Ran}(\text{pre}_D(P) \wedge \text{post}_D(Q)) \rrbracket_e b \longrightarrow (\llbracket (\forall na \cdot ((\#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n2 \rangle) \wedge (\#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n1 \rangle))) \rrbracket_e b)$ 
  by (simp add: forall-comb')
have n1-n2-eq:  $n2 = n1$ 
proof (rule ccontr)
  assume ss1:  $n2 \neq n1$ 
  have conj-false:  $(\forall na \cdot ((\#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n2 \rangle) \wedge (\#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n1 \rangle)))$ 
  = false
    using ss1 apply (rel-auto)
  done
have imp-false:  $\forall b. \llbracket \text{Ran}(\text{pre}_D(P) \wedge \text{post}_D(Q)) \rrbracket_e b \longrightarrow (\llbracket \text{false} \rrbracket_e b)$ 
  using P1-Q2-implies-n2-n1-2
  apply (simp add: conj-false)
done
have dom-false:  $\text{Ran}(\text{pre}_D(P) \wedge \text{post}_D(Q)) = \text{false}$ 
by (metis imp-false true-conj-zero(2) udeduct-refineI utp-pred-laws.inf.orderE utp-pred-laws.inf-commute)
have P1-Q2-false:  $(\text{pre}_D(P) \wedge \text{post}_D(Q)) = \text{false}$ 
  by (metis assume-Ran assume-false dom-false seqr-right-zero)
show False
  using s4 apply (simp add: P1-Q2-false)
done
qed
show ?thesis
  apply (simp add: n1-n2-eq m1-m2-eq)
done
qed

```

B.4 Operators

B.4.1 Id

```

lemma SimBlock-Id [simblock-healthy]:
  SimBlock 1 1 (Id)
  apply (simp add: f-sim-blocks)
  apply (rule SimBlock-FBlock)
  apply (simp add: f-blocks)
  apply (metis f-Const-def length-Cons list.size(3))
  by (simp add: f-blocks)

```

```

lemma inps-id: inps Id = 1
  using SimBlock-Id inps-outputs by auto

```

```

lemma outputs-id: outputs Id = 1
  using SimBlock-Id inps-outputs by auto

```

B.4.2 Sequential Composition

```

lemma refine-seq-mono:
  assumes  $P1 \sqsubseteq P2$  and  $Q1 \sqsubseteq Q2$ 
  shows  $P1 ; ; Q1 \sqsubseteq P2 ; ; Q2$ 
  by (simp add: assms(1) assms(2) seqr-mono)

```

```

lemma FBlock-seq-comp:

```

```

assumes s1: SimBlock m1 n1 (FBlock ( $\lambda x n. True$ ) m1 n1 f)
assumes s2: SimBlock n1 n2 (FBlock ( $\lambda x n. True$ ) n1 n2 g)
shows FBlock ( $\lambda x n. True$ ) m1 n1 f ;; FBlock ( $\lambda x n. True$ ) n1 n2 g = FBlock ( $\lambda x n. True$ ) m1 n2
( $g \circ f$ )
proof –
  show ?thesis
    apply (simp add: sim-blocks)
    apply (rel-simp)
    apply (rule iffI)
    apply (clarify)
    apply presburger
    apply (rel-auto)
  proof –
    fix okv inoutsv okv' inoutsv'
    assume a0: okv'
    assume a1: ( $\forall x. \text{length}(\text{inouts}_v x) = m1 \wedge \text{length}(\text{inouts}_{v'} x) = n2 \wedge$ 
       $g (f \text{inouts}_v) x = \text{inouts}_{v'} x$ )
    show  $\exists ok_{v''} \text{inouts}_{v''}$ .
      ( $ok_v \longrightarrow ok_{v''} \wedge (\forall x. \text{length}(\text{inouts}_{v''} x) = n1 \wedge f \text{inouts}_v x = \text{inouts}_{v''} x)$ 
         $\wedge (\forall x xa. \text{length}(x xa) = m1 \longrightarrow \text{length}(f x xa) = n1)) \wedge$ 
      ( $ok_{v''} \longrightarrow (\forall x. \text{length}(\text{inouts}_{v''} x) = n1 \wedge g \text{inouts}_{v''} x = \text{inouts}_{v'} x)$ 
         $\wedge (\forall x xa. \text{length}(x xa) = n1 \longrightarrow \text{length}(g x xa) = n2))$ 
    apply (rule-tac x = okv' in exI)
    apply (rule-tac x = f inoutsv in exI, simp)
    using SimBlock-FBlock-fn a0 a1 assms(2) s1 by blast
  qed
qed

```

```

lemma SimBlock-FBlock-seq-comp [simblock-healthy]:
assumes s1: SimBlock m1 n1 (FBlock ( $\lambda x n. True$ ) m1 n1 f)
assumes s2: SimBlock n1 n2 (FBlock ( $\lambda x n. True$ ) n1 n2 g)
shows SimBlock m1 n2 (FBlock ( $\lambda x n. True$ ) m1 n1 f ;; FBlock ( $\lambda x n. True$ ) n1 n2 g)
apply (simp add: s1 s2 FBlock-seq-comp)
apply (rule SimBlock-FBlock)
proof –
  obtain inoutsv::nat  $\Rightarrow$  real list where P:  $\forall na. \text{length}(\text{inouts}_v na) = m1$ 
  using list-len-avail by auto
  show  $\exists \text{inouts}_v \text{inouts}_{v'}. \forall x. \text{length}(\text{inouts}_{v'} x) = n2 \wedge \text{length}(\text{inouts}_v x) = m1 \wedge$ 
    ( $g \circ f$ ) inoutsv x = inoutsv' x
  apply (rule-tac x = inoutsv in exI)
  apply (rule-tac x = ( $g \circ f$ ) inoutsv in exI)
  using P SimBlock-FBlock-fn assms(2) s1 by auto
next
  show  $\forall x na. \text{length}(x na) = m1 \longrightarrow \text{length}((g \circ f) x na) = n2$ 
  using SimBlock-FBlock-fn assms(2) s1 by auto
qed

```

```

lemma FBlock-seq-comp':
assumes s1: SimBlock m1 n1 (FBlock (p1) m1 n1 f)
assumes s2: SimBlock n1 n2 (FBlock (p2) n1 n2 g)
shows FBlock ( $\lambda x n. p1 x n \wedge \text{length}(x n) = m1$ ) m1 n1 f ;;
  FBlock ( $\lambda x n. p2 x n \wedge \text{length}(x n) = n1$ ) n1 n2 g
  = FBlock ( $\lambda x n. p1 x n \wedge (p2 \circ f) x n \wedge \text{length}(x n) = m1$ ) m1 n2 ( $g \circ f$ )
proof –
  from s1 have 1:  $\forall x n. \text{length}(x n) = m1 \longrightarrow \text{length}(f x n) = n1$ 

```



```

    using SimBlock-FBlock-fn' by blast
  from s2 have 2:  $\forall x n. \text{length}(x n) = n1 \longrightarrow \text{length}(g x n) = n2$ 
    using SimBlock-FBlock-fn' by blast
  show ?thesis
    apply (simp add: sim-blocks)
    apply (simp add: ndesign-composition-wp wp-upred-def)
    apply (rule ref-eq)
    apply (rule ndesign-refine-intro)
    apply (rel-simp)
    using 1 apply fastforce
    apply (rel-simp)
    apply (rule-tac  $x = f \text{ inouts}_v$  in  $exI$ )
    using 1 2 apply simp
    apply (rule ndesign-refine-intro)
    apply (rel-simp)
    apply (metis ext)
    apply (rel-simp)
    by presburger
qed

```

lemma *SimBlock-FBlock-seq-comp'* [*simblock-healthy*]:

```

  assumes s1: SimBlock m1 n1 (FBlock (p1) m1 n1 f)
  assumes s2: SimBlock n1 n2 (FBlock (p2) n1 n2 g)

```

```

  assumes s3:  $\forall x n. (p1 x n) \longrightarrow (p2 \circ f) x n$ 

```

```

  shows SimBlock m1 n2 (FBlock ( $\lambda x n. p1 x n \wedge \text{length}(x n) = m1$ ) m1 n1 f ;;
    FBlock ( $\lambda x n. p2 x n \wedge \text{length}(x n) = n1$ ) n1 n2 g)

```

```

  apply (simp add: s1 s2 FBlock-seq-comp')

```

```

  apply (rule SimBlock-FBlock')

```

```

  proof -

```

```

    obtain  $\text{inouts}_v :: \text{nat} \Rightarrow \text{real list}$  where  $P: \forall na. \text{length}(\text{inouts}_v na) = m1 \wedge p1 \text{ inouts}_v na$ 
      using list-len-avail s1 SimBlock-FBlock-p by metis

```

```

    show  $\exists \text{inouts}_v.$ 

```

```

      ( $\forall x. p1 \text{ inouts}_v x \wedge p2 (f \text{ inouts}_v) x \wedge \text{length}(\text{inouts}_v x) = m1$ )  $\wedge$ 

```

```

      ( $\exists \text{inouts}'_v. \forall x. \text{length}(\text{inouts}'_v x) = n2 \wedge \text{length}(\text{inouts}_v x) = m1 \wedge (g \circ f) \text{ inouts}_v x = \text{inouts}'_v$ 

```

```

 $x$ )

```

```

    apply (rule-tac  $x = \text{inouts}_v$  in  $exI$ )

```

```

    apply (rule conjI)

```

```

    using P s3 apply auto[1]

```

```

    apply (rule-tac  $x = (g \circ f) \text{ inouts}_v$  in  $exI$ )

```

```

    using P assms(2) SimBlock-FBlock-fn' s1 by auto

```

```

  next

```

```

    show  $\forall x na. \text{length}(x na) = m1 \longrightarrow \text{length}((g \circ f) x na) = n2$ 

```

```

    using SimBlock-FBlock-fn' assms(2) s1 by auto

```

```

  qed

```

B.4.3 Parallel Composition

B.4.3.1 *mergeB* *ThreeWayMerge'*: similar to *ThreeWayMerge*, but it merges 1 and 2 firstly and then merges 0. Instead, *ThreeWayMerge* merges 0 and 1 firstly, then merges 2.

definition *ThreeWayMerge'* :: $'\alpha \text{ merge} \Rightarrow ((' \alpha, ' \alpha, (' \alpha, ' \alpha, ' \alpha) \text{ mrg}) \text{ mrg}, ' \alpha) \text{ urel } (\mathbf{M30}'(-))$ where [*upred-defs*]: $\text{ThreeWayMerge}' M = ((\$0 - \mathbf{v}' =_u \$0 - \mathbf{v} \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}_{<}) \wedge (\$0 - \mathbf{v}' =_u \$1 - 0 - \mathbf{v} \wedge \$1 - \mathbf{v}' =_u \$1 - 1 - \mathbf{v} \wedge \$\mathbf{v}_{<}' =_u \$\mathbf{v}_{<})) ; ; M ; ; U1 ; ; M$

mergeB is associative which means the order of merges applied to 0, 1 and 2 does not matter as

long as 0, 1, and 2 are merged in the same order. In other word, $M(M(0,1), 2) = M(0, M(1, 2))$

lemma *mergeB-assoc*: $ThreeWayMerge\ mergeB = ThreeWayMerge'\ mergeB$
apply (*simp add: ThreeWayMerge-def ThreeWayMerge'-def mergeB-def*)
apply (*rel-auto*)
apply (*rename-tac inouts_v0 ok_v0 inouts_v1 ok_v1 inouts_v2 ok_v2 inouts_v3 inouts_v4 inouts_v5 inouts_v6 inouts_v7*)
apply (*rule-tac x = (ok_v1 \wedge ok_v2) in exI*)
apply (*rule-tac x = λ na. (inouts_v2 na \bullet inouts_v3 na) in exI*)
apply (*simp*)
apply (*rule-tac x = λ na. (inouts_v2 na \bullet inouts_v3 na) in exI*)
apply (*simp*)
apply (*rename-tac inouts_v0 ok_v0 inouts_v1 ok_v1 inouts_v2 ok_v2 inouts_v3 inouts_v4 inouts_v5 inouts_v6*)
apply (*rule-tac x = inouts_v0 in exI*)
apply (*rule-tac x = (ok_v0 \wedge ok_v1) in exI*)
apply (*rule-tac x = λ na. (inouts_v1 na \bullet inouts_v2 na) in exI*)
apply (*simp*)
apply (*rule-tac x = λ na. (inouts_v1 na \bullet inouts_v2 na) in exI*)
apply (*simp*)
done

B.4.3.2 *sim-parallel lemma SimParallel-form*:

assumes *s1: SimBlock m1 n1 B1*
assumes *s2: SimBlock m2 n2 B2*
shows $(B1 \parallel_B B2) =$
 $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$
 $((\text{takem } (m1+m2) (m1)) ; ; B1) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D:inouts' \rrbracket \wedge$
 $((\text{dropm } (m1+m2) (m2)) ; ; B2) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D:inouts' \rrbracket \wedge$
 $(\forall n::nat \cdot (\$ \mathbf{v}_D:inouts' \langle\langle n \rangle\rangle)_a =_u (\langle\langle \text{append} \rangle\rangle (\langle\langle inouts_0 \rangle\rangle n)_a (\langle\langle inouts_1 \rangle\rangle n)_a))) \wedge$
 $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$
 $(\text{is ?lhs} = \text{?rhs})$
proof –
have *s3: inps B1 = m1*
using *s1 by (simp add: inps-outps)*
have *s4: inps B2 = m2*
using *s2 by (simp add: inps-outps)*
show *?thesis*
apply (*simp add: sim-parallel-def*)
apply (*simp add: s3 s4 mergeB-def*)
apply (*simp add: par-by-merge-alt-def, rel-auto*)
apply (*rename-tac ok_v inouts_v' inouts_v2 inouts_v3 ok_v3 inouts_v4 ok_v4 ok_v5 inouts_v5 inouts_v6 ok_v6 inouts_v7*)
apply *blast*
by *blast*
qed

lemma *SimBlock-parallel-pre-true [simblock-healthy]*:

assumes *s1: SimBlock m1 n1 (true \vdash_n Q1)*
assumes *s2: SimBlock m2 n2 (true \vdash_n Q2)*
shows $SimBlock (m1+m2) (n1+n2) ((true \vdash_n Q1) \parallel_B (true \vdash_n Q2))$
proof –
– 1. Simplify the parallel operation
have *1: ((true \vdash_n Q1) \parallel_B (true \vdash_n Q2)) =*
 $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$
 $((\text{takem } (m1+m2) (m1)) ; ; (true \vdash_n Q1)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D:inouts' \rrbracket \wedge$
 $((\text{dropm } (m1+m2) (m2)) ; ; (true \vdash_n Q2)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D:inouts' \rrbracket \wedge$

$(\forall n::\text{nat} \cdot (\$v_D:\text{inouts}' (\langle n \rangle)_a =_u (\langle\langle \text{append} \rangle (\langle \text{inouts}_0 n \rangle)_a (\langle \text{inouts}_1 n \rangle)_a))) \wedge$
 $(\$ok' =_u (\langle\langle ok_0 \rangle \wedge \langle\langle ok_1 \rangle \rangle)))$
using *SimParallel-form s1 s2 by auto*
— 2. Get some basic facts from assumptions
from *s1* **have** $Q1 \neq \text{false}$
 by (*simp add: SimBlock-def*)
then have $Q1\text{-not-false}: \exists \text{inouts}_v \text{inouts}'_v. \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v))$
 by (*rel-simp*)
from *s2* **have** $Q2 \neq \text{false}$
 by (*simp add: SimBlock-def*)
then have $Q2\text{-not-false}: \exists \text{inouts}_v \text{inouts}'_v. \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v))$
 by (*rel-simp*)
from *s1* **have** $(\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle m1 \rangle) \sqsubseteq \text{Dom}(\text{PrePost}((\text{true} \vdash_n Q1)))$
 by (*simp add: SimBlock-def*)
then have $\text{ref-m1}: \forall \text{inouts}_v \text{inouts}'_v x. \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = m1$
 by (*rel-simp*)
from *s2* **have** $(\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle m2 \rangle) \sqsubseteq \text{Dom}(\text{PrePost}((\text{true} \vdash_n Q2)))$
 by (*simp add: SimBlock-def*)
then have $\text{ref-m2}: \forall \text{inouts}_v \text{inouts}'_v x. \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v), (\text{inouts}_v = \text{inouts}'_v)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = m2$
 by (*rel-simp*)
have $(\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n1 \rangle) \sqsubseteq \text{Ran}(\text{PrePost}((\text{true} \vdash_n Q1)))$
 using *SimBlock-def s1 by auto*
then have $\text{ref-n1}: \forall \text{inouts}_v \text{inouts}'_v x. \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}'_v), (\text{inouts}_v = \text{inouts}_v)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = n1$
 by (*rel-simp*)
have $(\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n2 \rangle) \sqsubseteq \text{Ran}(\text{PrePost}((\text{true} \vdash_n Q2)))$
 using *SimBlock-def s2 by auto*
then have $\text{ref-n2}: \forall \text{inouts}_v \text{inouts}'_v x. \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}'_v), (\text{inouts}_v = \text{inouts}_v)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = n2$
 by (*rel-simp*)
— Subgoal 1 for *SimBlock-def*
have $c1: \text{PrePost}((\text{true} \vdash_n Q1) \parallel_B (\text{true} \vdash_n Q2)) \neq \text{false}$
 apply (*simp add: 1*)
 apply (*simp add: sim-blocks*)
 apply (*rel-auto*)
proof —
 obtain $\text{inouts}_v 1$ **and** $\text{inouts}'_v 1$ **and** $\text{inouts}_v 2$ **and** $\text{inouts}'_v 2$
 where $P1: \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v 1), (\text{inouts}_v = \text{inouts}'_v 1))$
 and $P2: \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}_v 2), (\text{inouts}_v = \text{inouts}'_v 2))$
 using $Q1\text{-not-false } Q2\text{-not-false}$ **by** *blast*
 show $\exists \text{inouts}_v \text{inouts}'_v.$
 $(\forall a \text{ aa } ab.$
 $(\exists ok_v. ok_v \wedge$
 $(\exists \text{inouts}'_v.$
 $(\forall x. (m1 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m2 \wedge \text{inouts}'_v x = []) \wedge$
 $(0 < m1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = m1 + m2 \wedge$
 $\text{length}(\text{inouts}'_v x) = m1 \wedge \text{take } m1 (\text{inouts}_v x) = \text{inouts}'_v x)) \wedge$
 $(ok_v \longrightarrow a \wedge \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}'_v), (\text{inouts}_v = ab)))))) \longrightarrow$
 $(\forall b. (\exists ok_v. ok_v \wedge$
 $(\exists \text{inouts}'_v.$
 $(\forall x. (m2 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m1 \wedge \text{inouts}'_v x = []) \wedge$
 $(0 < m2 \longrightarrow$

$$\begin{aligned}
& \text{length}(\text{inouts}_v x) = m1 + m2 \wedge \\
& \text{length}(\text{inouts}_{v'} x) = m2 \wedge \text{drop } m1 (\text{inouts}_v x) = \text{inouts}_{v'} x) \wedge \\
& (\text{ok}_v \longrightarrow aa \wedge \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}_{v'}), (\text{inouts}_v = b))) \longrightarrow \\
& (\exists x. \neg \text{inouts}_{v'} x = ab x \bullet b x) \vee a \wedge aa) \wedge \\
& (\exists a aa. (\exists \text{ok}_v. \text{ok}_v \wedge \\
& \quad (\exists \text{inouts}_{v}'. \\
& \quad (\forall x. (m1 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m2 \wedge \text{inouts}_{v'} x = []) \wedge \\
& \quad (0 < m1 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = m1 + m2 \wedge \\
& \quad \text{length}(\text{inouts}_{v'} x) = m1 \wedge \text{take } m1 (\text{inouts}_v x) = \text{inouts}_{v'} x)) \wedge \\
& \quad (\text{ok}_v \longrightarrow \llbracket Q1 \rrbracket_e ((\text{inouts}_v = \text{inouts}_{v'}), (\text{inouts}_v = aa)))) \wedge \\
& (\exists b. (\exists \text{ok}_v. \text{ok}_v \wedge \\
& \quad (\exists \text{inouts}_{v}'. \\
& \quad (\forall x. (m2 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m1 \wedge \text{inouts}_{v'} x = []) \wedge \\
& \quad (0 < m2 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = m1 + m2 \wedge \\
& \quad \text{length}(\text{inouts}_{v'} x) = m2 \wedge \text{drop } m1 (\text{inouts}_v x) = \text{inouts}_{v'} x)) \wedge \\
& \quad (\text{ok}_v \longrightarrow a \wedge \llbracket Q2 \rrbracket_e ((\text{inouts}_v = \text{inouts}_{v'}), (\text{inouts}_v = b)))) \wedge \\
& \quad (\forall x. \text{inouts}_{v'} x = aa x \bullet b x) \wedge a)) \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_v 1 na \bullet \text{inouts}_v 2 na \text{ in } exI) \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_{v'} 1 na \bullet \text{inouts}_{v'} 2 na \text{ in } exI) \\
& \text{apply (rule conjI)} \\
& \text{apply blast} \\
& \text{apply (rule-tac } x = \text{True in } exI) \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_{v'} 1 na \text{ in } exI) \\
& \text{apply (rule conjI)} \\
& \text{apply (rule-tac } x = \text{True in } exI) \\
& \text{apply (simp)} \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_v 1 na \text{ in } exI) \\
& \text{using } P1 P2 \text{ ref-m1 ref-m2 apply fastforce} \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_{v'} 2 na \text{ in } exI) \\
& \text{apply (simp)} \\
& \text{apply (rule-tac } x = \text{True in } exI) \\
& \text{apply (simp)} \\
& \text{apply (rule-tac } x = \lambda na. \text{inouts}_v 2 na \text{ in } exI) \\
& \text{using } P1 P2 \text{ ref-m1 ref-m2 by force} \\
& \text{qed} \\
& \text{— Subgoal 2 for } \textit{SimBlock-def} \\
& \text{have } c2: ((\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle m1+m2 \rangle) \sqsubseteq \text{Dom}(\text{PrePost}((\text{true} \vdash_n Q1) \parallel_B (\text{true} \\
& \vdash_n Q2)))) \\
& \text{apply (simp add: 1)} \\
& \text{apply (simp add: sim-blocks)} \\
& \text{apply (rel-simp)} \\
& \text{using } \textit{assms} \\
& \text{by (metis add.right-neutral not-gr-zero)} \\
& \text{— Subgoal 3 for } \textit{SimBlock-def} \\
& \text{have } c3: ((\forall na \cdot \#_u(\&\text{inouts}(\langle na \rangle)_a) =_u \langle n1+n2 \rangle) \sqsubseteq \text{Ran}(\text{PrePost}((\text{true} \vdash_n Q1) \parallel_B (\text{true} \vdash_n \\
& Q2)))) \\
& \text{apply (simp add: 1)} \\
& \text{apply (simp add: sim-blocks)} \\
& \text{apply (rel-simp)} \\
& \text{by (simp add: ref-n1 ref-n2)} \\
& \text{from } c1 c2 c3 \text{ show ?thesis} \\
& \text{apply (simp add: SimBlock-def)}
\end{aligned}$$

done
qed

Parallel composition of two SimBlocks (provided that the preconditions of both are condition) are still SimBlock.

lemma *SimBlock-parallel* [*simblock-healthy*]:

assumes *s1*: *SimBlock* *m1* *n1* ($P1 \vdash_n Q1$)

assumes *s2*: *SimBlock* *m2* *n2* ($P2 \vdash_n Q2$)

shows *SimBlock* (*m1+m2*) (*n1+n2*) ($(P1 \vdash_n Q1) \parallel_B (P2 \vdash_n Q2)$)

proof –

have *pform*: ($(P1 \vdash_n Q1) \parallel_B (P2 \vdash_n Q2)$) =

$(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$
 $((\text{takem } (m1+m2) (m1)) ; ; (P1 \vdash_n Q1)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$v_D: inouts' \rrbracket \wedge$
 $((\text{dropm } (m1+m2) (m2)) ; ; (P2 \vdash_n Q2)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$v_D: inouts' \rrbracket \wedge$
 $(\forall n::nat \cdot (\$v_D: inouts' (\langle\langle n \rangle\rangle)_a =_u (\langle\langle \text{append} \rangle\rangle (\langle\langle inouts_0 \rangle\rangle n)_a (\langle\langle inouts_1 \rangle\rangle n)_a))) \wedge$
 $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$

using *SimParallel-form s1 s2* **by** *auto*

– Subgoal 1 for *SimBlock-def*

have *c1*: *PrePost*($(P1 \vdash_n Q1) \parallel_B (P2 \vdash_n Q2)$) \neq *false*

apply (*simp add: pform*)

apply (*simp add: sim-blocks*)

apply (*rel-auto*)

proof –

obtain *inouts_v1*::*nat* \Rightarrow *real list* **and** *inouts_v'1*::*nat* \Rightarrow *real list* **and**
inouts_v2::*nat* \Rightarrow *real list* **and** *inouts_v'2*::*nat* \Rightarrow *real list* **where**

P1: $\llbracket P1 \rrbracket_e (\langle\langle inouts_v = inouts_v1 \rangle\rangle)$ **and**

Q1: $\llbracket Q1 \rrbracket_e (\langle\langle inouts_v = inouts_v1 \rangle\rangle, \langle\langle inouts_v = inouts_v'1 \rangle\rangle)$ **and**

P2: $\llbracket P2 \rrbracket_e (\langle\langle inouts_v = inouts_v2 \rangle\rangle)$ **and**

Q2: $\llbracket Q2 \rrbracket_e (\langle\langle inouts_v = inouts_v2 \rangle\rangle, \langle\langle inouts_v = inouts_v'2 \rangle\rangle)$

using *s1 s2 SimBlock-implies-not-PQ'*

by *blast*

have *inps1*: *length*(*inouts_v1* *na*) = *m1*

using *P1 Q1 SimBlock-implies-mP s1* **by** *blast*

have *inps2*: *length*(*inouts_v2* *na*) = *m2*

using *P2 Q2 SimBlock-implies-mP s2* **by** *blast*

have *outps1*: *length*(*inouts_v'1* *na*) = *n1*

using *P1 Q1 SimBlock-implies-Qn s1* **by** *blast*

have *outps2*: *length*(*inouts_v'2* *na*) = *n2*

using *P2 Q2 SimBlock-implies-Qn s2* **by** *blast*

show $\exists inouts_v inouts_v'$.

($\forall a aa ab.$

$(\exists ok_v. ok_v \wedge$

$(\exists inouts_v'.$

$(\forall x. (m1 = 0 \longrightarrow \text{length}(inouts_v x) = m2 \wedge inouts_v' x = []) \wedge$

$(0 < m1 \longrightarrow$

$\text{length}(inouts_v x) = m1 + m2 \wedge$

$\text{length}(inouts_v' x) = m1 \wedge \text{take } m1 (inouts_v x) = inouts_v' x)) \wedge$

$(ok_v \wedge \llbracket P1 \rrbracket_e (\langle\langle inouts_v = inouts_v' \rangle\rangle) \longrightarrow$

$a \wedge \llbracket Q1 \rrbracket_e (\langle\langle inouts_v = inouts_v' \rangle\rangle, \langle\langle inouts_v = ab \rangle\rangle))) \longrightarrow$

$(\forall b. (\exists ok_v. ok_v \wedge$

$(\exists inouts_v'.$

$(\forall x. (m2 = 0 \longrightarrow \text{length}(inouts_v x) = m1 \wedge inouts_v' x = []) \wedge$

$(0 < m2 \longrightarrow$

$\text{length}(inouts_v x) = m1 + m2 \wedge$

$\text{length}(inouts_v' x) = m2 \wedge \text{drop } m1 (inouts_v x) = inouts_v' x)) \wedge$

$$\begin{aligned}
& (ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v' | \!) \longrightarrow \\
& \quad aa \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v' | \!, (\!| inouts_v = b | \!))) \longrightarrow \\
(\exists x. \neg inouts_v' x = ab x \bullet b x) \vee a \wedge aa) \wedge \\
(\exists a \text{ aa. } (\exists ok_v. ok_v \wedge \\
\quad (\exists inouts_v'. \\
\quad (\forall x. (m1 = 0 \longrightarrow length(inouts_v x) = m2 \wedge inouts_v' x = []) \wedge \\
\quad (0 < m1 \longrightarrow \\
\quad \quad length(inouts_v x) = m1 + m2 \wedge \\
\quad \quad length(inouts_v' x) = m1 \wedge take\ m1\ (inouts_v x) = inouts_v' x)) \wedge \\
\quad (ok_v \wedge \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v' | \!) \longrightarrow \\
\quad \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v' | \!, (\!| inouts_v = aa | \!)))) \wedge \\
(\exists b. (\exists ok_v. ok_v \wedge \\
\quad (\exists inouts_v'. \\
\quad (\forall x. (m2 = 0 \longrightarrow length(inouts_v x) = m1 \wedge inouts_v' x = []) \wedge \\
\quad (0 < m2 \longrightarrow \\
\quad \quad length(inouts_v x) = m1 + m2 \wedge \\
\quad \quad length(inouts_v' x) = m2 \wedge drop\ m1\ (inouts_v x) = inouts_v' x)) \wedge \\
\quad (ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v' | \!) \longrightarrow \\
\quad \quad a \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v' | \!, (\!| inouts_v = b | \!)))) \wedge \\
\quad (\forall x. inouts_v' x = aa x \bullet b x) \wedge a)) \\
\text{apply (rule-tac } x = \lambda na . (inouts_v1\ na \bullet inouts_v2\ na) \text{ in } exI) \\
\text{apply (rule-tac } x = \lambda na . (inouts_v'1\ na \bullet inouts_v'2\ na) \text{ in } exI) \\
\text{apply (rule conjI) \\
apply (rule allI)+ \\
apply (simp) \\
apply (rule impI) \\
apply (rule allI)+ \\
apply (rule impI) \\
proof - \\
\text{fix } ok_{v1} \text{ and } ok_{v2} \text{ and } inouts_{v1}'::nat \Rightarrow \text{real list and } inouts_{v2}'::nat \Rightarrow \text{real list} \\
\text{assume } a1: \exists ok_v. ok_v \wedge \\
(\exists inouts_v'. \\
(\forall x. (m1 = 0 \longrightarrow length(inouts_{v1}\ x) + length(inouts_{v2}\ x) = m2 \wedge inouts_v' x = []) \wedge \\
(0 < m1 \longrightarrow \\
length(inouts_{v1}\ x) + length(inouts_{v2}\ x) = m1 + m2 \wedge \\
length(inouts_v' x) = m1 \wedge \\
take\ m1\ (inouts_{v1}\ x) \bullet take\ (m1 - length(inouts_{v1}\ x))\ (inouts_{v2}\ x) = \\
inouts_v' x)) \wedge \\
(ok_v \wedge \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v' | \!) \longrightarrow \\
ok_{v1} \wedge \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v' | \!, (\!| inouts_v = inouts_{v1}' | \!)))) \\
\text{assume } a2: \exists ok_v. ok_v \wedge \\
(\exists inouts_v'. \\
(\forall x. (m2 = 0 \longrightarrow length(inouts_{v1}\ x) + length(inouts_{v2}\ x) = m1 \wedge inouts_v' x = []) \wedge \\
(0 < m2 \longrightarrow \\
length(inouts_{v1}\ x) + length(inouts_{v2}\ x) = m1 + m2 \wedge \\
length(inouts_v' x) = m2 \wedge \\
drop\ m1\ (inouts_{v1}\ x) \bullet drop\ (m1 - length(inouts_{v1}\ x))\ (inouts_{v2}\ x) = \\
inouts_v' x)) \wedge \\
(ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v' | \!) \longrightarrow \\
ok_{v2} \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v' | \!, (\!| inouts_v = inouts_{v2}' | \!)))) \\
\text{from } a1 \text{ have } 1: \exists ok_v. ok_v \wedge \\
(\exists inouts_v'. \\
(\forall x. (m1 = 0 \longrightarrow \\
length(inouts_{v1}\ x) + length(inouts_{v2}\ x) = m2 \wedge \\
inouts_{v1}\ x = [] \wedge
\end{aligned}$$

$inouts_v' x = [] \wedge$
 $(0 < m1 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m1 + m2 \wedge$
 $length(inouts_v' x) = m1 \wedge$
 $inouts_v1 x = inouts_v' x) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket)) \rightarrow$
 $ok_v1 \wedge \llbracket Q1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket, \llbracket inouts_v = inouts_v1' \rrbracket))$
using *inps1 P1 Q1 SimBlock-implies-mP s1*
by (*smt append-take-drop-id cancel-comm-monoid-add-class.diff-cancel length-0-conv*
length-drop take-eq-Nil)
then have 2: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $(\forall x. inouts_v1 x = inouts_v' x \wedge$
 $(m1 = 0 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m2 \wedge$
 $inouts_v1 x = [] \wedge$
 $(0 < m1 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m1 + m2 \wedge$
 $length(inouts_v1 x) = m1)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket)) \rightarrow$
 $ok_v1 \wedge \llbracket Q1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket, \llbracket inouts_v = inouts_v1' \rrbracket))$)
by (*metis (full-types) inps1 length-0-conv length-greater-0-conv*)
then have 3: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $(\forall x. inouts_v1 x = inouts_v' x) \wedge$
 $(\forall x. (m1 = 0 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m2 \wedge$
 $inouts_v1 x = [] \wedge$
 $(0 < m1 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m1 + m2 \wedge$
 $length(inouts_v1 x) = m1)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket)) \rightarrow$
 $ok_v1 \wedge \llbracket Q1 \rrbracket_e (\llbracket inouts_v = inouts_v' \rrbracket, \llbracket inouts_v = inouts_v1' \rrbracket))$)
by *smt*
then have 4: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $(\forall x. (m1 = 0 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m2 \wedge$
 $inouts_v1 x = [] \wedge$
 $(0 < m1 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m1 + m2 \wedge$
 $length(inouts_v1 x) = m1)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\llbracket inouts_v = inouts_v1 \rrbracket)) \rightarrow$
 $ok_v1 \wedge \llbracket Q1 \rrbracket_e (\llbracket inouts_v = inouts_v1 \rrbracket, \llbracket inouts_v = inouts_v1' \rrbracket))$)
by (*metis 2 3 append-Nil ext length-append less-not-refl neq0-conv*)
then have 5: $\exists ok_v. ok_v \wedge$
 $(\forall x. (m1 = 0 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m2 \wedge$
 $inouts_v1 x = [] \wedge$
 $(0 < m1 \rightarrow$
 $length(inouts_v1 x) + length(inouts_v2 x) = m1 + m2 \wedge$
 $length(inouts_v1 x) = m1)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\llbracket inouts_v = inouts_v1 \rrbracket)) \rightarrow$
 $ok_v1 \wedge \llbracket Q1 \rrbracket_e (\llbracket inouts_v = inouts_v1 \rrbracket, \llbracket inouts_v = inouts_v1' \rrbracket))$)
by (*simp*)

then have 6:
 $(\forall x. (m1 = 0 \longrightarrow$
 $\quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m2 \wedge$
 $\quad inouts_v 1 x = [])) \wedge$
 $(0 < m1 \longrightarrow$
 $\quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 + m2 \wedge$
 $\quad \text{length}(inouts_v 1 x) = m1)) \wedge$
 $(\llbracket P1 \rrbracket_e (\!(inouts_v = inouts_v 1)\!) \longrightarrow$
 $\quad ok_v 1 \wedge \llbracket Q1 \rrbracket_e (\!(inouts_v = inouts_v 1)\!), (\!(inouts_v = inouts_v 1'\!))$

by blast
then have 7: $(\llbracket P1 \rrbracket_e (\!(inouts_v = inouts_v 1)\!) \longrightarrow$
 $\quad ok_v 1 \wedge \llbracket Q1 \rrbracket_e (\!(inouts_v = inouts_v 1)\!), (\!(inouts_v = inouts_v 1'\!))$

by simp
from a2 have 11: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $\quad (\forall x. (m2 = 0 \longrightarrow \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 \wedge$
 $\quad \quad inouts_v' x = [] \wedge inouts_v 2 x = [])) \wedge$
 $\quad (0 < m2 \longrightarrow$
 $\quad \quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 + m2 \wedge$
 $\quad \quad \text{length}(inouts_v' x) = m2 \wedge$
 $\quad \quad (inouts_v 2 x) = inouts_v' x)) \wedge$
 $\quad (ok_v \wedge \llbracket P2 \rrbracket_e (\!(inouts_v = inouts_v'\!)) \longrightarrow$
 $\quad \quad ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!(inouts_v = inouts_v'\!), (\!(inouts_v = inouts_v 2'\!))))$

using inps1 P2 Q2 SimBlock-implies-mP s2
by (smt P1 Q1 append-self-conv2 cancel-comm-monoid-add-class.diff-cancel drop-0
 $\quad \text{drop-eq-Nil order-refl s1)$

then have 12: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $\quad (\forall x. inouts_v 2 x = inouts_v' x \wedge$
 $\quad \quad (m2 = 0 \longrightarrow \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 \wedge$
 $\quad \quad \quad inouts_v 2 x = [])) \wedge$
 $\quad \quad (0 < m2 \longrightarrow$
 $\quad \quad \quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 + m2 \wedge$
 $\quad \quad \quad \text{length}(inouts_v 2 x) = m2)) \wedge$
 $\quad \quad (ok_v \wedge \llbracket P2 \rrbracket_e (\!(inouts_v = inouts_v'\!)) \longrightarrow$
 $\quad \quad \quad ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!(inouts_v = inouts_v'\!), (\!(inouts_v = inouts_v 2'\!))))$

by (metis (full-types) inps2 length-0-conv length-greater-0-conv)

then have 13: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $\quad (\forall x. inouts_v 2 x = inouts_v' x) \wedge$
 $\quad \quad (\forall x. (m2 = 0 \longrightarrow \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 \wedge$
 $\quad \quad \quad inouts_v 2 x = [])) \wedge$
 $\quad \quad (0 < m2 \longrightarrow$
 $\quad \quad \quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 + m2 \wedge$
 $\quad \quad \quad \text{length}(inouts_v 2 x) = m2)) \wedge$
 $\quad \quad (ok_v \wedge \llbracket P2 \rrbracket_e (\!(inouts_v = inouts_v'\!)) \longrightarrow$
 $\quad \quad \quad ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!(inouts_v = inouts_v'\!), (\!(inouts_v = inouts_v 2'\!))))$

by smt
then have 14: $\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $\quad (\forall x. (m2 = 0 \longrightarrow \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 \wedge$
 $\quad \quad \quad inouts_v 2 x = [])) \wedge$
 $\quad \quad (0 < m2 \longrightarrow$
 $\quad \quad \quad \text{length}(inouts_v 1 x) + \text{length}(inouts_v 2 x) = m1 + m2 \wedge$
 $\quad \quad \quad \text{length}(inouts_v 2 x) = m2)) \wedge$

$(ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!)) \longrightarrow$
 $ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 |, (\!| inouts_v = inouts_v 2' | \!))$
by (*metis 12 13 append-Nil ext length-append less-not-refl neq0-conv*)
then have 15: $\exists ok_v. ok_v \wedge$
 $(\forall x. (m2 = 0 \longrightarrow length(inouts_v 1 x) + length(inouts_v 2 x) = m1 \wedge$
 $inouts_v 2 x = [])) \wedge$
 $(0 < m2 \longrightarrow$
 $length(inouts_v 1 x) + length(inouts_v 2 x) = m1 + m2 \wedge$
 $length(inouts_v 2 x) = m2)) \wedge$
 $(ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!)) \longrightarrow$
 $ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 |, (\!| inouts_v = inouts_v 2' | \!))$
by (*simp*)
then have 16:
 $(\forall x. (m2 = 0 \longrightarrow length(inouts_v 1 x) + length(inouts_v 2 x) = m1 \wedge$
 $inouts_v 2 x = [])) \wedge$
 $(0 < m2 \longrightarrow$
 $length(inouts_v 1 x) + length(inouts_v 2 x) = m1 + m2 \wedge$
 $length(inouts_v 2 x) = m2)) \wedge$
 $(\llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!)) \longrightarrow$
 $ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 |, (\!| inouts_v = inouts_v 2' | \!))$
by *blast*
then have 17: $(\llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!)) \longrightarrow$
 $ok_v 2 \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 |, (\!| inouts_v = inouts_v 2' | \!))$
by *simp*
show $(\exists x. \neg inouts_v' 1 x \bullet inouts_v' 2 x = inouts_v 1' x \bullet inouts_v 2' x) \vee ok_v 1 \wedge ok_v 2$
proof (*rule ccontr*)
assume *aa:* $\neg ((\exists x. \neg inouts_v' 1 x \bullet inouts_v' 2 x = inouts_v 1' x \bullet inouts_v 2' x) \vee ok_v 1 \wedge$
 $ok_v 2)$
from *aa* **have** *b1:* $(\forall x. inouts_v' 1 x \bullet inouts_v' 2 x = inouts_v 1' x \bullet inouts_v 2' x) \wedge (\neg ok_v 1$
 $\vee \neg ok_v 2)$
by (*simp*)
from *b1* **have** *b2:* $(\forall x. inouts_v' 1 x \bullet inouts_v' 2 x = inouts_v 1' x \bullet inouts_v 2' x)$
by (*simp*)
from *b1* **have** *b3:* $(\neg ok_v 1 \vee \neg ok_v 2)$
by (*simp*)
from *b3* **7** *17* **have** *b4:*
 $\neg \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!)) \vee$
 $\neg \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!))$
by *blast*
from *s1* **have** *b5:* $\llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!))$
using *P1 SimBlock-implies-not-P-cond*
by *blast*
from *s2* **have** *b6:* $\llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!))$
using *P2 SimBlock-implies-not-P-cond* **by** *blast*
show *False*
using *b4 b5 b6* **by** (*auto*)
qed
next
show $\exists a aa. (\exists ok_v. ok_v \wedge$
 $(\exists inouts_v'.$
 $(\forall x. (m1 = 0 \longrightarrow length(inouts_v 1 x \bullet inouts_v 2 x) = m2 \wedge inouts_v' x = [])) \wedge$
 $(0 < m1 \longrightarrow$
 $length(inouts_v 1 x \bullet inouts_v 2 x) = m1 + m2 \wedge$
 $length(inouts_v' x) = m1 \wedge take m1 (inouts_v 1 x \bullet inouts_v 2 x) = inouts_v' x)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v' | \!)) \longrightarrow$

$$\llbracket Q1 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v' \rangle \rangle, \langle \langle \text{inouts}_v = aa \rangle \rangle)) \wedge$$

$$(\exists b. (\exists ok_v. ok_v \wedge$$

$$(\exists \text{inouts}_v'.$$

$$(\forall x. (m2 = 0 \longrightarrow \text{length}(\text{inouts}_v1\ x \bullet \text{inouts}_v2\ x) = m1 \wedge \text{inouts}_v' x = []) \wedge$$

$$(0 < m2 \longrightarrow$$

$$\text{length}(\text{inouts}_v1\ x \bullet \text{inouts}_v2\ x) = m1 + m2 \wedge$$

$$\text{length}(\text{inouts}_v' x) = m2 \wedge \text{drop } m1 (\text{inouts}_v1\ x \bullet \text{inouts}_v2\ x) = \text{inouts}_v' x)) \wedge$$

$$(\text{ok}_v \wedge \llbracket P2 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v' \rangle \rangle) \longrightarrow$$

$$a \wedge \llbracket Q2 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v' \rangle \rangle, \langle \langle \text{inouts}_v = b \rangle \rangle)) \wedge$$

$$(\forall x. \text{inouts}_v'1\ x \bullet \text{inouts}_v'2\ x = aa\ x \bullet b\ x) \wedge a)$$
apply (*rule-tac* $x = \text{True}$ **in** exI)
apply (*rule-tac* $x = \text{inouts}_v'1$ **in** exI)
apply (*rule conjI*)
apply (*rule-tac* $x = \text{True}$ **in** exI , *simp*)
apply (*rule-tac* $x = \text{inouts}_v1$ **in** exI)
using $P1\ P2\ Q1\ Q2\ \text{SimBlock-implies-mP}\ s1\ s2$
apply (*smt add-eq-self-zero append.right-neutral*
cancel-ab-semigroup-add-class.add-diff-cancel-left' order-refl sum-eq-sum-conv
take-all take-eq-Nil)
apply (*rule-tac* $x = \text{inouts}_v'2$ **in** exI , *simp*)
apply (*rule-tac* $x = \text{True}$ **in** exI , *simp*)
apply (*rule-tac* $x = \text{inouts}_v2$ **in** exI)
using $P1\ P2\ Q1\ Q2\ \text{SimBlock-implies-mP}\ s1\ s2$
by (*smt add-eq-self-zero append-eq-append-conv-if*
cancel-ab-semigroup-add-class.add-diff-cancel-left' drop-0 list-exhaust-size-eq0
sum-eq-sum-conv)

qed

qed

— Subgoal 2 for *SimBlock-def*

have $c2: ((\forall na \cdot \#_u(\&\text{inouts}(\langle \langle na \rangle \rangle)_a) =_u \langle \langle m1+m2 \rangle \rangle) \sqsubseteq \text{Dom}(\text{PrePost}((P1 \vdash_n Q1) \parallel_B (P2 \vdash_n Q2))))$

apply (*simp add: pform*)
apply (*simp add: sim-blocks*)
apply (*rel-simp*)
using *assms*
by (*metis add.right-neutral not-gr-zero*)

— Subgoal 3 for *SimBlock-def*

have $c3: ((\forall na \cdot \#_u(\&\text{inouts}(\langle \langle na \rangle \rangle)_a) =_u \langle \langle n1+n2 \rangle \rangle) \sqsubseteq \text{Ran}(\text{PrePost}((P1 \vdash_n Q1) \parallel_B (P2 \vdash_n Q2))))$

apply (*simp add: pform*)
apply (*simp add: sim-blocks*)
apply (*rel-simp*)
apply (*rename-tac inouts_v' inouts_v n ok_v q1 ok_v q2 inouts_v1' ok_v inouts_v2' inouts_v1 ok_v' inouts_v2*)
proof —
fix $\text{inouts}_v' \text{ inouts}_v\ n\ \text{ok}_v\ q1\ \text{ok}_v\ q2\ \text{inouts}_v1'\ \text{ok}_v\ \text{inouts}_v2'\ \text{inouts}_v1\ \text{ok}_v' \text{ inouts}_v2$
assume $a1: \llbracket P1 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v1 \rangle \rangle) \longrightarrow \llbracket Q1 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v1 \rangle \rangle, \langle \langle \text{inouts}_v = \text{inouts}_v1' \rangle \rangle)$
assume $a2: \llbracket P2 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v2 \rangle \rangle) \longrightarrow \llbracket Q2 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v2 \rangle \rangle, \langle \langle \text{inouts}_v = \text{inouts}_v2' \rangle \rangle)$
assume $a3: \forall a\ aa\ ab.$
 $(\exists ok_v. ok_v \wedge$
 $(\exists \text{inouts}_v.$
 $(\forall x. (m1 = 0 \longrightarrow \text{inouts}_v\ x = []) \wedge$
 $(0 < m1 \longrightarrow \text{length}(\text{inouts}_v\ x) = m1 \wedge \text{inouts}_v1\ x = \text{inouts}_v\ x)) \wedge$
 $(\text{ok}_v \wedge \llbracket P1 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v \rangle \rangle) \longrightarrow$
 $a \wedge \llbracket Q1 \rrbracket_e (\langle \langle \text{inouts}_v = \text{inouts}_v \rangle \rangle, \langle \langle \text{inouts}_v = ab \rangle \rangle)) \longrightarrow$

$(\forall b. (\exists ok_v. ok_v \wedge$
 $(\exists inouts_v.$
 $(\forall x. (m2 = 0 \longrightarrow inouts_v x = []) \wedge$
 $(0 < m2 \longrightarrow length(inouts_v x) = m2 \wedge inouts_v 2 x = inouts_v x)) \wedge$
 $(ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v | \!) \longrightarrow$
 $aa \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v | \!), (\!| inouts_v = b | \!))) \longrightarrow$
 $(\exists x. \neg inouts_v 1' x \bullet inouts_v 2' x = ab x \bullet b x) \vee a \wedge aa)$
assume $a_4: \forall x. inouts_v' x = inouts_v 1' x \bullet inouts_v 2' x$
assume $a_5: \forall x. (m1 = 0 \longrightarrow length(inouts_v x) = m2 \wedge inouts_v 1 x = []) \wedge$
 $(0 < m1 \longrightarrow length(inouts_v x) = m1 + m2 \wedge length(inouts_v 1 x) = m1 \wedge$
 $take\ m1\ (inouts_v\ x) = inouts_v 1\ x)$
assume $a_6: \forall x. (m2 = 0 \longrightarrow length(inouts_v x) = m1 \wedge inouts_v 2 x = []) \wedge$
 $(0 < m2 \longrightarrow length(inouts_v x) = m1 + m2 \wedge length(inouts_v 2 x) = m2 \wedge$
 $drop\ m1\ (inouts_v\ x) = inouts_v 2\ x)$
from a_5 **have** $1: length(inouts_v 1 na) = m1$
by *blast*
from a_6 **have** $2: length(inouts_v 2 na) = m2$
by *blast*
from a_3 **have** $(\forall a\ aa\ ab.$
 $(\exists ok_v. ok_v \wedge$
 $(\exists inouts_v.$
 $(\forall x. (m1 = 0 \longrightarrow inouts_v x = []) \wedge$
 $(0 < m1 \longrightarrow length(inouts_v x) = m1 \wedge inouts_v 1 x = inouts_v x)) \wedge$
 $(ok_v \wedge \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v | \!) \longrightarrow$
 $a \wedge \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v | \!), (\!| inouts_v = ab | \!))) \longrightarrow$
 $(\forall b. (\exists ok_v. ok_v \wedge$
 $(\exists inouts_v.$
 $(\forall x. (m2 = 0 \longrightarrow inouts_v x = []) \wedge$
 $(0 < m2 \longrightarrow length(inouts_v x) = m2 \wedge inouts_v 2 x = inouts_v x)) \wedge$
 $(ok_v \wedge \llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v | \!) \longrightarrow$
 $aa \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v | \!), (\!| inouts_v = b | \!))) \longrightarrow$
 $(\exists x. \neg inouts_v 1' x \bullet inouts_v 2' x = ab x \bullet b x) \vee a \wedge aa)$
 $\longrightarrow (\forall a\ aa\ ab.$
 $(\llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!) \longrightarrow$
 $a \wedge \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!), (\!| inouts_v = ab | \!)) \longrightarrow$
 $(\forall b. (\llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!) \longrightarrow$
 $aa \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!), (\!| inouts_v = b | \!)) \longrightarrow$
 $(\exists x. \neg inouts_v 1' x \bullet inouts_v 2' x = ab x \bullet b x) \vee a \wedge aa)$
apply (*simp*)
apply (*rule allI*)
apply (*rename-tac* $ok_v q\ inouts_v 1' q\ inouts_v 2' q$)
apply (*rule impI*)
apply (*rule allI*)
apply (*rule impI*)
by (*smt* $a_5\ a_6\ neq0\ conv$)
then **have** $a_3': (\forall a\ aa\ ab.$
 $(\llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!) \longrightarrow$
 $a \wedge \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!), (\!| inouts_v = ab | \!)) \longrightarrow$
 $(\forall b. (\llbracket P2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!) \longrightarrow$
 $aa \wedge \llbracket Q2 \rrbracket_e (\!| inouts_v = inouts_v 2 | \!), (\!| inouts_v = b | \!)) \longrightarrow$
 $(\exists x. \neg inouts_v 1' x \bullet inouts_v 2' x = ab x \bullet b x) \vee a \wedge aa)$
using a_3 **by** *smt*
have $P1: \llbracket P1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!)$
using a_3' **using** a_2 **by** *blast*
then **have** $Q1: \llbracket Q1 \rrbracket_e (\!| inouts_v = inouts_v 1 | \!), (\!| inouts_v = inouts_v 1' | \!)$

```

    using a1 by auto
  then have N1: length(inouts_v 1' n) = n1
    using P1 SimBlock-implies-Qn s1 by blast
  have P2:  $\llbracket P2 \rrbracket_e$  ( $\langle inouts_v = inouts_v 2 \rangle$ )
    using a3' using a1 by blast
  then have Q2:  $\llbracket Q2 \rrbracket_e$  ( $\langle inouts_v = inouts_v 2 \rangle$ , ( $\langle inouts_v = inouts_v 2' \rangle$ ))
    using a2 by auto
  then have N2: length(inouts_v 2' n) = n2
    using P2 SimBlock-implies-Qn s2 by blast
  show length(inouts_v 1' n) + length(inouts_v 2' n) = n1 + n2
    using N1 N2 by auto
qed
from c1 c2 c3 show ?thesis
  apply (simp add: SimBlock-def)
done
qed

```

lemma inps-parallel:

```

  assumes s1: SimBlock m1 n1 (P1  $\vdash_n$  Q1)
  assumes s2: SimBlock m2 n2 (P2  $\vdash_n$  Q2)
  shows inps ((P1  $\vdash_n$  Q1)  $\parallel_B$  (P2  $\vdash_n$  Q2)) = m1 + m2
  using SimBlock-parallel inps-outps s1 s2 by blast

```

lemma outps-parallel:

```

  assumes s1: SimBlock m1 n1 (P1  $\vdash_n$  Q1)
  assumes s2: SimBlock m2 n2 (P2  $\vdash_n$  Q2)
  shows outps ((P1  $\vdash_n$  Q1)  $\parallel_B$  (P2  $\vdash_n$  Q2)) = n1 + n2
  using SimBlock-parallel inps-outps
  using s1 s2 by blast

```

Associativity of parallel composition.

lemma parallel-ass:

```

  assumes s1: SimBlock m0 n0 (P0  $\vdash_n$  Q0)
  assumes s2: SimBlock m1 n1 (P1  $\vdash_n$  Q1)
  assumes s3: SimBlock m2 n2 (P2  $\vdash_n$  Q2)
  shows ((P0  $\vdash_n$  Q0)  $\parallel_B$  ((P1  $\vdash_n$  Q1)  $\parallel_B$  (P2  $\vdash_n$  Q2))) = (((P0  $\vdash_n$  Q0)  $\parallel_B$  (P1  $\vdash_n$  Q1))  $\parallel_B$  (P2  $\vdash_n$  Q2))
  (is ?lhs = ?rhs)

```

proof –

```

  let ?P12 =  $\exists$  (ok1, ok2, inouts1, inouts2)  $\cdot$ 
    (((takem (m1+m2) (m1)) ; ; (P1  $\vdash_n$  Q1))  $\llbracket \langle ok_1 \rangle, \langle inouts_1 \rangle / \$ok', \$v_D: inouts' \rrbracket$   $\wedge$ 
    ((dropm (m1+m2) (m2)) ; ; (P2  $\vdash_n$  Q2))  $\llbracket \langle ok_2 \rangle, \langle inouts_2 \rangle / \$ok', \$v_D: inouts' \rrbracket$   $\wedge$ 
    ( $\forall n::nat \cdot (\$v_D: inouts' \langle n \rangle_a =_u \langle \langle append \rangle \langle inouts_1 n \rangle_a \langle inouts_2 n \rangle_a \rangle) \wedge$ 
    ( $\$ok' =_u \langle \langle ok_1 \rangle \wedge \langle ok_2 \rangle \rangle$ )))

```

```

  have lhs-12: ((P1  $\vdash_n$  Q1)  $\parallel_B$  (P2  $\vdash_n$  Q2)) = ?P12

```

```

  using SimParallel-form s2 s3 by blast

```

```

  have lhs-12-sim: SimBlock (m1+m2) (n1+n2) ((P1  $\vdash_n$  Q1)  $\parallel_B$  (P2  $\vdash_n$  Q2))

```

```

  by (simp add: SimBlock-parallel s2 s3)

```

```

  then have lhs-sim: ?lhs =

```

```

    ( $\exists$  (ok0, ok12, inouts0, inouts12)  $\cdot$ 
    (((takem (m0+(m1+m2)) (m0)) ; ; (P0  $\vdash_n$  Q0))  $\llbracket \langle ok_0 \rangle, \langle inouts_0 \rangle / \$ok', \$v_D: inouts' \rrbracket$   $\wedge$ 
    ((dropm (m0+(m1+m2)) (m1+m2)) ; ; ?P12)  $\llbracket \langle ok_{12} \rangle, \langle inouts_{12} \rangle / \$ok', \$v_D: inouts' \rrbracket$   $\wedge$ 
    ( $\forall n::nat \cdot (\$v_D: inouts' \langle n \rangle_a =_u \langle \langle append \rangle \langle inouts_0 n \rangle_a \langle inouts_{12} n \rangle_a \rangle) \wedge$ 
    ( $\$ok' =_u \langle \langle ok_0 \rangle \wedge \langle ok_{12} \rangle \rangle$ )))

```

```

  using lhs-12-sim lhs-12 SimParallel-form s1 s2 s3 by auto

```

```

let ?P01 =  $\exists$  (ok0, ok1, inouts0, inouts1) ·
  (((takem (m0+m1) (m0)) ;; (P0 ⊢n Q0))[[«ok0», «inouts0»/$ok', $vD:inouts']] ∧
  ((dropm (m0+m1) (m1)) ;; (P1 ⊢n Q1))[[«ok1», «inouts1»/$ok', $vD:inouts']] ∧
  (∀ n::nat · ($vD:inouts' («n») a =u («append» («inouts0 n») a («inouts1 n») a))) ∧
  ($ok' =u («ok0» ∧ «ok1»)))
have rhs-01: ((P0 ⊢n Q0) ||B (P1 ⊢n Q1)) = ?P01
  using SimParallel-form s1 s2 by blast
have rhs-01-sim: SimBlock (m0+m1) (n0+n1) ((P0 ⊢n Q0) ||B (P1 ⊢n Q1))
  by (simp add: SimBlock-parallel s1 s2)
then have rhs-sim: ?rhs =
  ( $\exists$  (ok01, ok2, inouts01, inouts2) ·
    (((takem ((m0+m1)+m2) (m0+m1)) ;; ?P01)[[«ok01», «inouts01»/$ok', $vD:inouts']] ∧
    ((dropm ((m0+m1)+m2) (m2)) ;; (P2 ⊢n Q2))[[«ok2», «inouts2»/$ok', $vD:inouts']] ∧
    (∀ n::nat · ($vD:inouts' («n») a =u («append» («inouts01 n») a («inouts2 n») a))) ∧
    ($ok' =u («ok01» ∧ «ok2»))))
  using rhs-01-sim rhs-01 SimParallel-form s1 s2 s3 by auto
show ?thesis
  apply (simp add: lhs-sim rhs-sim)
  apply (simp add: sim-blocks)
  apply (rel-simp)
  apply (rule iffI)
  — Subgoal 1: lhs → rhs
  apply (clarify)
  apply (rename-tac okv inoutsv okv' inoutsv' okv'q0 aa inoutsv'q0 okvp0 inoutsv'12 inoutsvp0
okv12
    inoutsv12 okv'q1 okv'q2 inoutsv'q1 okvp1 inoutsv'q2 inoutsvp1 okvp2 inoutsvp2)
  apply (rule-tac x = okv'q0 ∧ okv'q1 in exI)
  apply (rule-tac x = okv'q2 in exI)
  apply (rule-tac x = λna. (inoutsv'q0 na • inoutsv'q1 na) in exI)
  apply (rule conjI)
  apply (rule-tac x = okv in exI)
  apply (rule-tac x = λna. (inoutsvp0 na • inoutsvp1 na) in exI)
  apply (rule conjI)
  apply (clarify)
  apply (smt ab-semigroup-add-class.add-ac(1) drop-0 gr0I length-append list.size(3)
self-append-conv take-add)
  apply (rule-tac x = okv'q0 in exI)
  apply (rule-tac x = okv'q1 in exI)
  apply (rule-tac x = inoutsv'q0 in exI)
  apply (rule conjI)
  apply (rule-tac x = okvp0 in exI)
  apply (rule-tac x = inoutsvp0 in exI)
  apply (rule conjI, simp)
  apply (metis gr0I length-0-conv)
  apply blast
  apply (rule-tac x = inoutsv'q1 in exI)
  apply (rule conjI)
  apply (rule-tac x = okvp1 in exI)
  apply (rule-tac x = inoutsvp1 in exI)
  apply (rule conjI, simp)
  apply (metis append-eq-conv-conj drop-append list.size(3) neq0-conv)
  apply blast
  apply blast
  apply (rule-tac x = inoutsv'q2 in exI)

```

```

apply (rule conjI, simp)
apply (rule-tac x = okvp2 in exI)
apply (rule-tac x = inoutsvp2 in exI)
apply (rule conjI, simp)
apply (metis add-cancel-left-right drop-drop gr0I semiring-normalization-rules(24))
apply blast
apply auto[1]
— Subgoal 2: rhs -> lhs
apply (clarify)
  apply (rename-tac okv inoutsv okv' inoutsv' a okv'q2 inoutsv'01 okv01 inoutsv'q2 inoutsv01
okvp2 inoutsvp2
  okv'q0 okv'q1 inoutsv'q0 okvp0 inoutsv'q1 inoutsvp0 okvp1 inoutsvp1)
apply (rule-tac x = okv'q0 in exI)
apply (rule-tac x = okv'q1 ∧ okv'q2 in exI)
apply (rule-tac x = λna. (inoutsv'q0 na) in exI)
apply (rule conjI)
apply (rule-tac x = okv in exI)
apply (rule-tac x = λna. (inoutsvp0 na) in exI)
apply (rule conjI, simp)
apply (rule impI)
apply (rule allI)
apply (rule conjI)
apply (metis add-cancel-left-left zero-less-iff-neq-zero)
apply (metis append.right-neutral append-take-drop-id diff-is-0-eq le-add1 take-0 take-append)
apply blast
apply (rule-tac x = λna. (inoutsv'q1 na • inoutsv'q2 na) in exI)
apply (rule conjI)
apply (rule-tac x = okv in exI)
apply (rule-tac x = λna. (inoutsvp1 na • inoutsvp2 na) in exI)
apply (rule conjI, simp)
apply (rule impI)
apply (rule allI)
apply (rule conjI)
apply (smt add.commute append-take-drop-id drop-drop length-append length-greater-0-conv
less-add-same-cancel2 neq0-conv take-drop)
apply (rule impI)
apply (rule conjI)
apply (metis gr-zeroI list.size(3))
apply (metis (no-types, hide-lams) add.left-neutral append-take-drop-id diff-add-zero drop-0
drop-append neq0-conv plus-list-def zero-list-def)
apply (rule-tac x = okv'q1 in exI)
apply (rule-tac x = okv'q2 in exI)
apply (rule-tac x = inoutsv'q1 in exI)
apply (rule conjI, simp)
apply (metis gr0I length-0-conv)
apply (rule-tac x = inoutsv'q2 in exI)
apply (rule conjI)
apply (rule-tac x = okvp2 in exI)
apply (rule-tac x = inoutsvp2 in exI)
apply (rule conjI, simp)
apply (metis append-eq-conv-conj drop-append list.size(3) neq0-conv)
apply blast
apply blast
apply (rule conjI, simp)
by blast

```

qed

lemma refinement-implies-r:

assumes $s1: (P1 \vdash_r Q1) \sqsubseteq (P1r \vdash_r Q1r)$

shows $\forall ok_v inouts_v ok_v' inouts_v'$.

$$\begin{aligned} & (ok_v \wedge \llbracket P1r \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v'))) \longrightarrow \\ & \quad ok_v' \wedge \llbracket Q1r \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v')) \longrightarrow \\ & (ok_v \wedge \llbracket P1 \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v'))) \longrightarrow \\ & \quad ok_v' \wedge \llbracket Q1 \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v')) \end{aligned}$$

using $s1$ **apply** (*rel-simp*)

by *blast*

lemma refinement-implies:

assumes $s1: (P1 \vdash_n Q1) \sqsubseteq (P1r \vdash_n Q1r)$

shows $\forall ok_v inouts_v ok_v' inouts_v'$.

$$\begin{aligned} & (ok_v \wedge \llbracket P1r \rrbracket_e ((\downarrow inouts_v = inouts_v)) \longrightarrow \\ & \quad ok_v' \wedge \llbracket Q1r \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v'))) \longrightarrow \\ & (ok_v \wedge \llbracket P1 \rrbracket_e ((\downarrow inouts_v = inouts_v)) \longrightarrow \\ & \quad ok_v' \wedge \llbracket Q1 \rrbracket_e ((\downarrow inouts_v = inouts_v), (\downarrow inouts_v = inouts_v'))) \end{aligned}$$

using $s1$ **apply** (*rel-simp*)

by *blast*

lemma parallel-mono-r:

assumes $s1: SimBlock\ m1\ n1\ (P1 \vdash_r Q1)$

assumes $s2: SimBlock\ m2\ n2\ (P2 \vdash_r Q2)$

assumes $s3: SimBlock\ m1\ n1\ (P1r \vdash_r Q1r)$

assumes $s4: SimBlock\ m2\ n2\ (P2r \vdash_r Q2r)$

assumes $s5: (P1 \vdash_r Q1) \sqsubseteq (P1r \vdash_r Q1r)$

assumes $s6: (P2 \vdash_r Q2) \sqsubseteq (P2r \vdash_r Q2r)$

shows $((P1 \vdash_r Q1) \parallel_B (P2 \vdash_r Q2)) \sqsubseteq ((P1r \vdash_r Q1r) \parallel_B (P2r \vdash_r Q2r))$

proof –

have $pform: ((P1 \vdash_r Q1) \parallel_B (P2 \vdash_r Q2)) =$

$$\begin{aligned} & (\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot \\ & \quad (((takem\ (m1+m2)\ (m1)) \ ; \ ; \ (P1 \vdash_r Q1)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D: inouts' \rrbracket \wedge \\ & \quad ((dropm\ (m1+m2)\ (m2)) \ ; \ ; \ (P2 \vdash_r Q2)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D: inouts' \rrbracket \wedge \\ & \quad (\forall n::nat \cdot (\$ \mathbf{v}_D: inouts' (\langle\langle n \rangle\rangle)_a =_u (\langle\langle append \rangle\rangle (\langle\langle inouts_0 \ n \rangle\rangle)_a (\langle\langle inouts_1 \ n \rangle\rangle)_a))) \wedge \\ & \quad (\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))) \end{aligned}$$

using *SimParallel-form s1 s2* **by** *auto*

have $pform': ((P1r \vdash_r Q1r) \parallel_B (P2r \vdash_r Q2r)) =$

$$\begin{aligned} & (\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot \\ & \quad (((takem\ (m1+m2)\ (m1)) \ ; \ ; \ (P1r \vdash_r Q1r)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D: inouts' \rrbracket \wedge \\ & \quad ((dropm\ (m1+m2)\ (m2)) \ ; \ ; \ (P2r \vdash_r Q2r)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D: inouts' \rrbracket \wedge \\ & \quad (\forall n::nat \cdot (\$ \mathbf{v}_D: inouts' (\langle\langle n \rangle\rangle)_a =_u (\langle\langle append \rangle\rangle (\langle\langle inouts_0 \ n \rangle\rangle)_a (\langle\langle inouts_1 \ n \rangle\rangle)_a))) \wedge \\ & \quad (\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))) \end{aligned}$$

using *SimParallel-form s3 s4* **by** *auto*

show *?thesis*

apply (*simp add: pform pform'*)

apply (*simp add: sim-blocks*)

apply (*rel-simp*)

apply (*rename-tac ok_v inouts_v inouts_v' ok_v q1r ok_v q2r inouts_v 1r' ok_v p1r inouts_v 2r' inouts_v 1r ok_v p2r inouts_v 2r*)

apply (*rule-tac x = ok_v q1r in exI*)

apply (*rule-tac x = ok_v q2r in exI*)

apply (*rule-tac x = inouts_v 1r' in exI*)

```

apply (simp)
apply (rule conjI)
apply (rule-tac x = okvp1r in exI, simp)
apply (rule-tac x = inoutsv1r in exI)
apply (rule conjI)
apply simp
using s5 s1 refinement-implies-r apply (metis)
apply (rule-tac x = inoutsv2r' in exI, simp)
apply (rule-tac x = okvp2r in exI)
apply simp
apply (rule-tac x = inoutsv2r in exI, simp)
using s6 s2 refinement-implies-r apply (metis)
done
qed

```

lemma parallel-mono:

```

assumes s1: SimBlock m1 n1 (P1 ⊢n Q1)
assumes s2: SimBlock m2 n2 (P2 ⊢n Q2)
assumes s3: SimBlock m1 n1 (P1r ⊢n Q1r)
assumes s4: SimBlock m2 n2 (P2r ⊢n Q2r)
assumes s5: (P1 ⊢n Q1) ⊆ (P1r ⊢n Q1r)
assumes s6: (P2 ⊢n Q2) ⊆ (P2r ⊢n Q2r)
shows  $((P1 ⊢_n Q1) \parallel_B (P2 ⊢_n Q2)) \subseteq ((P1r ⊢_n Q1r) \parallel_B (P2r ⊢_n Q2r))$ 
proof –
  have pform: ((P1 ⊢n Q1) ∥B (P2 ⊢n Q2)) =
     $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$ 
       $((\text{takem } (m1+m2) (m1)) ; ; (P1 \vdash_n Q1)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$v_D:inouts' \rrbracket \wedge$ 
       $((\text{dropm } (m1+m2) (m2)) ; ; (P2 \vdash_n Q2)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$v_D:inouts' \rrbracket \wedge$ 
       $(\forall n::nat \cdot (\$v_D:inouts' (\langle\langle n \rangle\rangle)_a =_u (\langle\langle \text{append} \rangle\rangle (\langle\langle inouts_0 \rangle\rangle n)_a (\langle\langle inouts_1 \rangle\rangle n)_a))) \wedge$ 
       $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$ 
    )
    using SimParallel-form s1 s2 by auto
  have pform': ((P1r ⊢n Q1r) ∥B (P2r ⊢n Q2r)) =
     $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$ 
       $((\text{takem } (m1+m2) (m1)) ; ; (P1r \vdash_n Q1r)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$v_D:inouts' \rrbracket \wedge$ 
       $((\text{dropm } (m1+m2) (m2)) ; ; (P2r \vdash_n Q2r)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$v_D:inouts' \rrbracket \wedge$ 
       $(\forall n::nat \cdot (\$v_D:inouts' (\langle\langle n \rangle\rangle)_a =_u (\langle\langle \text{append} \rangle\rangle (\langle\langle inouts_0 \rangle\rangle n)_a (\langle\langle inouts_1 \rangle\rangle n)_a))) \wedge$ 
       $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$ 
    )
    using SimParallel-form s3 s4 by auto
  show ?thesis
  apply (simp add: pform pform')
  apply (simp add: sim-blocks)
  apply (rel-simp)
  apply (rename-tac okv inoutsv inoutsv' okvq1r okvq2r inoutsv1r' okvp1r inoutsv2r' inoutsv1r
okvp2r inoutsv2r))
  apply (rule-tac x = okvq1r in exI)
  apply (rule-tac x = okvq2r in exI)
  apply (rule-tac x = inoutsv1r' in exI)
  apply (simp)
  apply (rule conjI)
  apply (rule-tac x = okvp1r in exI, simp)
  apply (rule-tac x = inoutsv1r in exI)
  apply (rule conjI)
  apply simp
  using s5 s1 refinement-implies apply (metis)
  apply (rule-tac x = inoutsv2r' in exI, simp)

```



```

apply (rule-tac x = okvp2r in exI)
apply simp
apply (rule-tac x = inoutsv2r in exI, simp)
using s6 s2 refinement-implies apply (metis)
done
qed

```

lemma *FBlock-parallel-comp-id*:

```

assumes s1: SimBlock 1 1 (FBlock (λx n. True) 1 1 f-Id)
shows (FBlock (λx n. True) 1 1 f-Id) ||B (FBlock (λx n. True) 1 1 f-Id)
  = FBlock (λx n. True) 2 2 (λx n. (((f-Id ∘ (λxx nn. take 1 (xx nn))) x n)
    • ((f-Id ∘ (λxx nn. drop 1 (xx nn)))) x n))

```

proof –

```

have inps-1: inps (FBlock (λx n. True) (Suc 0) (Suc 0) f-Id) = 1

```

```

  using s1 by (simp add: inps-P)

```

```

have form: ((FBlock (λx n. True) 1 1 f-Id) ||B (FBlock (λx n. True) 1 1 f-Id)) =

```

```

  (∃ (ok0, ok1, inouts0, inouts1) •

```

```

    (((takem (1+1) (1)) ;; (FBlock (λx n. True) 1 1 f-Id)) [«ok0», «inouts0»/$ok', $vD:inouts'])

```

∧

```

    ((dropm (1+1) (1)) ;; (FBlock (λx n. True) 1 1 f-Id)) [«ok1», «inouts1»/$ok', $vD:inouts'])

```

∧

```

    (∀ n::nat • ($vD:inouts' («n»)a =u («append» («inouts0 n»)a («inouts1 n»)a))) ∧
    ($ok' =u («ok0» ∧ «ok1»)))

```

```

  using s1 by (simp add: SimParallel-form)

```

```

have 2: (∃ (ok0, ok1, inouts0, inouts1) •

```

```

    (((takem (1+1) (1)) ;; (FBlock (λx n. True) 1 1 f-Id)) [«ok0», «inouts0»/$ok', $vD:inouts'])

```

∧

```

    ((dropm (1+1) (1)) ;; (FBlock (λx n. True) 1 1 f-Id)) [«ok1», «inouts1»/$ok', $vD:inouts'])

```

∧

```

    (∀ n::nat • ($vD:inouts' («n»)a =u («append» («inouts0 n»)a («inouts1 n»)a))) ∧
    ($ok' =u («ok0» ∧ «ok1»)))

```

```

  = FBlock (λx n. True) 2 2 (λx n. (((f-Id ∘ (λxx nn. take 1 (xx nn))) x n)

```

```

    • ((f-Id ∘ (λxx nn. drop 1 (xx nn)))) x n))

```

```

apply (simp add: FBlock-def f-Id-def takem-def dropm-def)

```

```

apply (rel-auto)

```

```

apply (simp add: f-Id-def)

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = inoutsv' in exI)

```

```

apply (rule conjI)

```

```

apply blast

```

```

apply (rule-tac x = λna. [] in exI)

```

```

apply blast

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = λna. take (Suc 0) (inoutsv na) in exI)

```

```

apply (rule conjI)

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = λna. take (Suc 0) (inoutsv na) in exI)

```

```

apply (metis (no-types, lifting) Nitpick.size-list-simp(2) f-Id-def less-numeral-extra(3)

```

```

  list.sel(1) pos2 take-Suc take-eq-Nil take-tl)

```

```

apply (rule-tac x = λna. drop (Suc 0) (inoutsv na) in exI)

```

```

apply (rule conjI)

```

```

apply (rule-tac x = okv' in exI)

```

```

apply (rule-tac x = λna. drop (Suc 0) (inoutsv na) in exI)

```

apply (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-le-mono diff-Suc-1*
drop-eq-Nil f-Id-def hd-drop-conv-nth le-numeral-extra(4) length-drop lessI numeral-2-eq-2)
by (*metis Cons-nth-drop-Suc Suc-1 Suc-eq-plus1 add.left-neutral append-take-drop-id drop-0*
drop-eq-Nil lessI list.sel(1) order-refl take-Suc zero-less-Suc)
show *?thesis*
using *form 2*
by *simp*
qed

lemma *FBlock-parallel-comp:*

assumes *s1: SimBlock m1 n1 (FBlock (λx n. True) m1 n1 f)*
assumes *s2: SimBlock m2 n2 (FBlock (λx n. True) m2 n2 g)*
shows $(FBlock (\lambda x n. True) m1 n1 f) \parallel_B (FBlock (\lambda x n. True) m2 n2 g)$
 $= FBlock (\lambda x n. True) (m1+m2) (n1+n2)$
 $(\lambda x n. (((f \circ (\lambda xx nn. take m1 (xx nn))) x n) \bullet ((g \circ (\lambda xx nn. drop m1 (xx nn)))) x n))$
proof –
have *inps-1: inps (FBlock (λx n. True) m1 n1 f) = m1*
using *s1 by (simp add: inps-P)*
have *inps-2: inps (FBlock (λx n. True) m2 n2 g) = m2*
using *s2 by (simp add: inps-P)*
have form: $((FBlock (\lambda x n. True) m1 n1 f) \parallel_B (FBlock (\lambda x n. True) m2 n2 g)) =$
 $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$
 $((takem (m1+m2) (m1)) ; ; (FBlock (\lambda x n. True) m1 n1 f)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D : inouts' \rrbracket$
 \wedge
 $((dropm (m1+m2) (m2)) ; ; (FBlock (\lambda x n. True) m2 n2 g)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D : inouts' \rrbracket$
 \wedge
 $(\forall n::nat \cdot (\$ \mathbf{v}_D : inouts' \langle\langle n \rangle\rangle)_a =_u (\langle\langle append \rangle\rangle (\langle\langle inouts_0 n \rangle\rangle)_a (\langle\langle inouts_1 n \rangle\rangle)_a)) \wedge$
 $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$
using *s1 s2 by (simp add: SimParallel-form)*
have 2: $(\exists (ok_0, ok_1, inouts_0, inouts_1) \cdot$
 $((takem (m1+m2) (m1)) ; ; (FBlock (\lambda x n. True) m1 n1 f)) \llbracket \langle\langle ok_0 \rangle\rangle, \langle\langle inouts_0 \rangle\rangle / \$ok', \$\mathbf{v}_D : inouts' \rrbracket$
 \wedge
 $((dropm (m1+m2) (m2)) ; ; (FBlock (\lambda x n. True) m2 n2 g)) \llbracket \langle\langle ok_1 \rangle\rangle, \langle\langle inouts_1 \rangle\rangle / \$ok', \$\mathbf{v}_D : inouts' \rrbracket$
 \wedge
 $(\forall n::nat \cdot (\$ \mathbf{v}_D : inouts' \langle\langle n \rangle\rangle)_a =_u (\langle\langle append \rangle\rangle (\langle\langle inouts_0 n \rangle\rangle)_a (\langle\langle inouts_1 n \rangle\rangle)_a)) \wedge$
 $(\$ok' =_u (\langle\langle ok_0 \rangle\rangle \wedge \langle\langle ok_1 \rangle\rangle)))$
 $= FBlock (\lambda x n. True) (m1+m2) (n1+n2)$
 $(\lambda x n. (((f \circ (\lambda xx nn. take m1 (xx nn))) x n) \bullet ((g \circ (\lambda xx nn. drop m1 (xx nn)))) x n))$
apply (*simp add: FBlock-def f-Id-def takem-def dropm-def*)
apply (*rel-simp*)
apply (*rule iffI*)
apply (*clarify*)
apply (*rule conjI, simp*)
apply (*rule conjI, simp*)
proof –
fix *ok_v inouts_v inouts_v' a aa ab ok_v'' b inouts_v''':nat ⇒ real list and ok_v''' and*
inouts_v''':nat ⇒ real list
assume *a1: ∀x. (m1 = 0 → length(inouts_v x) = m2 ∧ inouts_v'' x = []) ∧*
 $(0 < m1 \longrightarrow \text{length}(inouts_v x) = m1 + m2 \wedge \text{take } m1 (inouts_v x) = inouts_v'' x)$
assume *a2: ∀x. (m2 = 0 → length(inouts_v x) = m1 ∧ inouts_v''' x = []) ∧*
 $(0 < m2 \longrightarrow \text{length}(inouts_v x) = m1 + m2 \wedge \text{drop } m1 (inouts_v x) = inouts_v''' x)$
assume *a3: ∀x. length(inouts_v'' x) = m1 ∧ length(ab x) = n1 ∧ f inouts_v'' x = ab x*
assume *a4: ∀x. length(inouts_v''' x) = m2 ∧ length(b x) = n2 ∧ g inouts_v''' x = b x*
from a1 have 1: ∀x. take m1 (inouts_v x) = inouts_v'' x
by fastforce

then have $11: \text{inouts}_v'' = (\lambda x. \text{take } m1 (\text{inouts}_v x))$
using $a1$ **by** *force*
from $a3$ **have** $2: \forall x. f \text{ inouts}_v'' x = ab x$
by *blast*
from 11 **and** 2 **have** $3: \forall x. f (\lambda x. \text{take } m1 (\text{inouts}_v x)) x = ab x$
by *blast*
from $a2$ **have** $g1: \forall x. (\text{drop } m1 (\text{inouts}_v x) = \text{inouts}_v''' x)$
by *fastforce*
then have $g11: \text{inouts}_v''' = (\lambda x. \text{drop } m1 (\text{inouts}_v x))$
by *force*
from $a4$ **have** $g2: \forall x. g \text{ inouts}_v''' x = b x$
by *blast*
from $g11$ **and** $g2$ **have** $g3: \forall x. g (\lambda x. \text{drop } m1 (\text{inouts}_v x)) x = b x$
by *blast*
show $\forall x. \text{length}(\text{inouts}_v x) = m1 + m2 \wedge$
 $f (\lambda nn. \text{take } m1 (\text{inouts}_v nn)) x \bullet g (\lambda nn. \text{drop } m1 (\text{inouts}_v nn)) x = ab x \bullet b x$
apply (*rule allI*)
apply (*rule conjI*)
using $a2$ **apply** *auto*[1]
by (*simp add: 3 g3*)
next
assume $a1: \forall x xa. \text{length}(x xa) = m1 \longrightarrow \text{length}(f x xa) = n1$
assume $a2: \forall x xa. \text{length}(x xa) = m2 \longrightarrow \text{length}(g x xa) = n2$
show $\forall x xa. \text{length}(x xa) = m1 + m2 \longrightarrow$
 $\text{length}(f (\lambda nn. \text{take } m1 (x nn)) xa) + \text{length}(g (\lambda nn. \text{drop } m1 (x nn)) xa) = n1 + n2$
using $a1 a2$ **by** *simp*
next
fix $ok_v \text{ inouts}_v ok_v' \text{ inouts}_v'$
assume $a1: ok_v \longrightarrow$
 $ok_v' \wedge$
 $(\forall x. \text{length}(\text{inouts}_v x) = m1 + m2 \wedge$
 $\text{length}(\text{inouts}_v' x) = n1 + n2 \wedge$
 $f (\lambda nn. \text{take } m1 (\text{inouts}_v nn)) x \bullet g (\lambda nn. \text{drop } m1 (\text{inouts}_v nn)) x = \text{inouts}_v' x) \wedge$
 $(\forall x xa. \text{length}(x xa) = m1 + m2 \longrightarrow$
 $\text{length}(f (\lambda nn. \text{take } m1 (x nn)) xa) + \text{length}(g (\lambda nn. \text{drop } m1 (x nn)) xa) = n1 + n2)$
from $a1$ **show** $\exists a aa ab.$
 $(\exists ok_v' \text{ inouts}_v'.$
 $(ok_v \longrightarrow$
 $ok_v' \wedge$
 $(\forall x. (m1 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m2 \wedge \text{inouts}_v' x = []) \wedge$
 $(0 < m1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = m1 + m2 \wedge \text{length}(\text{inouts}_v' x) = m1 \wedge \text{take } m1 (\text{inouts}_v x) =$
 $\text{inouts}_v' x))) \wedge$
 $(ok_v' \longrightarrow$
 $a \wedge (\forall x. \text{length}(\text{inouts}_v' x) = m1 \wedge \text{length}(ab x) = n1 \wedge f \text{ inouts}_v' x = ab x) \wedge$
 $(\forall x xa. \text{length}(x xa) = m1 \longrightarrow \text{length}(f x xa) = n1))) \wedge$
 $(\exists b. (\exists ok_v' \text{ inouts}_v'.$
 $(ok_v \longrightarrow$
 $ok_v' \wedge$
 $(\forall x. (m2 = 0 \longrightarrow \text{length}(\text{inouts}_v x) = m1 \wedge \text{inouts}_v' x = []) \wedge$
 $(0 < m2 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = m1 + m2 \wedge$
 $\text{length}(\text{inouts}_v' x) = m2 \wedge \text{drop } m1 (\text{inouts}_v x) = \text{inouts}_v' x))) \wedge$
 $(ok_v' \longrightarrow$
 $aa \wedge (\forall x. \text{length}(\text{inouts}_v' x) = m2 \wedge \text{length}(b x) = n2 \wedge g \text{ inouts}_v' x = b x) \wedge$

```

      (∀ x xa. length(x xa) = m2 → length(g x xa) = n2))) ∧
      (∀ x. inouts_v' x = ab x • b x) ∧ ok_v' = (a ∧ aa))
apply (rel-auto)
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = inouts_v' in exI)
apply (rule conjI)
apply blast
using take-0 apply blast
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = λna. f (λnx. take m1 (inouts_v nx)) na in exI)
apply (rule conjI)
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = λnx. take m1 (inouts_v nx) in exI)
using SimBlock-FBlock-fn s1 apply auto[1]
apply (rule-tac x = λna. g (λnx. drop m1 (inouts_v nx)) na in exI)
apply (rule conjI)
apply (rule-tac x = ok_v' in exI)
apply (rule-tac x = λnx. drop m1 (inouts_v nx) in exI)
using SimBlock-FBlock-fn s2 apply auto[1]
by simp
qed
show ?thesis
using 2 form by simp
qed

```

lemma *SimBlock-FBlock-parallel-comp* [*simblock-healthy*]:

```

assumes s1: SimBlock m1 n1 (FBlock (λx n. True) m1 n1 f)
assumes s2: SimBlock m2 n2 (FBlock (λx n. True) m2 n2 g)
shows SimBlock (m1+m2) (n1+n2) ((FBlock (λx n. True) m1 n1 f) ||B (FBlock (λx n. True) m2
n2 g))
apply (simp add: s1 s2 FBlock-parallel-comp)
apply (rule SimBlock-FBlock)
proof –
obtain inouts_v::nat ⇒ real list where P: ∀ na. length(inouts_v na) = m1 + m2
using list-len-avail by auto
show ∃ inouts_v inouts_v'.
  ∀ x. length(inouts_v' x) = n1 + n2 ∧
  length(inouts_v x) = m1 + m2 ∧
  f (λnn. take m1 (inouts_v nn)) x • g (λnn. drop m1 (inouts_v nn)) x = inouts_v' x
apply (rule-tac x = inouts_v in exI)
apply (rule-tac x = λna. (f (λnn. take m1 (inouts_v nn)) na • g (λnn. drop m1 (inouts_v nn))
na) in exI)
using P SimBlock-FBlock-fn s1 s2 by auto
next
show ∀ x na. length(x na) = m1 + m2 →
  length(f (λnn. take m1 (x nn)) na • g (λnn. drop m1 (x nn)) na) = n1 + n2
using SimBlock-FBlock-fn s1 s2 by auto
qed

```

B.4.4 Feedback

B.4.4.1 feedback lemma *feedback-mono*:

```

fixes m1 :: nat and n1 :: nat and i1 :: nat and o1 :: nat
assumes s1: SimBlock m1 n1 P1

```

```

assumes s2: SimBlock m1 n1 P2
assumes s3: P1  $\sqsubseteq$  P2
assumes s4: i1 < m1
assumes s5: o1 < n1
shows (P1 fD (i1,o1))  $\sqsubseteq$  (P2 fD (i1,o1))
apply (simp add: f-sim-blocks)
using s1 s2 apply (simp add: inps-P outps-P)
apply (rel-simp)
apply (auto)
apply (metis s3 upred-ref-iff)
apply (rule-tac x = x in exI)
apply (rule-tac x = okv'' in exI)
apply (rule-tac x = inoutsv'' in exI)
apply (rule-tac x = okv''' in exI)
apply (rule-tac x = inoutsv''' in exI)
apply (metis s3 upred-ref-iff)
apply (rule-tac x = x in exI)
apply (rule-tac x = True in exI)
apply (rule-tac x = inoutsv'' in exI)
apply (rule conjI)
apply blast
apply (rule-tac x = False in exI)
apply (rule-tac x = inoutsv''' in exI)
apply (meson s3 upred-ref-iff)
apply (rule-tac x = x in exI)
apply (rule-tac x = True in exI)
apply (rule-tac x = inoutsv'' in exI)
apply (rule conjI)
apply blast
apply (rule-tac x = okv''' in exI)
apply (rule-tac x = inoutsv''' in exI)
by (metis s3 upred-ref-iff)

```

lemma *sol-f-id: Solvable 0 0 1 1 f-Id*
by (*simp add: Solvable-def f-Id-def f-PreFD-def*)

lemma *sol-f-ud: Solvable 0 0 1 1 (f-UnitDelay x0)*
apply (*simp add: Solvable-def f-UnitDelay-def f-PreFD-def*)
by (*auto*)

— The function which output is equal to its input plus 1 is not solvable

lemma \neg *Solvable 0 0 1 1 ($\lambda x n. [hd(x n) + 1]$)*
apply (*simp add: Solvable-def f-PreFD-def*)
by (*auto*)

lemma *sol-f-id-ud: Solvable 0 0 1 1 ((f-UnitDelay x0) \circ (f-Id))*
apply (*simp add: Solvable-def f-UnitDelay-def f-Id-def f-PreFD-def*)
by (*auto*)

lemma *sol-f-integrator*:

Solvable 1 1 2 2 ($\lambda x n. [if\ n = 0\ then\ x0\ else\ (x\ (n-1)!0) + (x\ (n-1)!1),$
 $if\ n = 0\ then\ x0\ else\ (x\ (n-1)!0) + (x\ (n-1)!1)]$)
apply (*simp add: Solvable-def f-PreFD-def*)
apply (*clarify*)
apply (*rule-tac x = $\lambda na. (if\ na = 0\ then\ x0\ else\ (x0 + sum-hd-signal\ inouts_0\ (na-1))$ in exI*)
apply (*simp, clarify*)
apply (*rule conjI*)
apply (*clarify*)
apply (*metis Nil-is-append-conv One-nat-def add commute hd-append2 hd-conv-nth list.size(3)*
 $nth-append-length\ zero-neg-one$)
apply (*clarify*)
proof –
fix $inouts_0::nat \Rightarrow real\ list$ **and** $n::nat$
assume $a1: \forall x. length(inouts_0\ x) = Suc\ 0$
assume $a2: \neg n \leq Suc\ 0$
have $1: (inouts_0\ (n - Suc\ 0)) \bullet [x0 + sum-hd-signal\ inouts_0\ (n - Suc\ (Suc\ 0))](0)$
 $= hd(inouts_0\ (n - Suc\ 0))$
using $a1\ a2$
by (*metis One-nat-def hd-conv-nth le-numeral-extra(4) less-numeral-extra(1) list.size(3)*
 $not-one-le-zero\ nth-append$)
have $2: (inouts_0\ (n - Suc\ 0)) \bullet [x0 + sum-hd-signal\ inouts_0\ (n - Suc\ (Suc\ 0))](Suc\ 0)$
 $= x0 + sum-hd-signal\ inouts_0\ (n - Suc\ (Suc\ 0))$
using $a1\ a2$
by (*metis nth-append-length*)
have $3: (n - (Suc\ 0)) = Suc\ (n - (Suc\ (Suc\ 0)))$
using $a2$ **by** *linarith*
show $x0 + sum-hd-signal\ inouts_0\ (n - Suc\ 0) =$
 $(inouts_0\ (n - Suc\ 0)) \bullet [x0 + sum-hd-signal\ inouts_0\ (n - Suc\ (Suc\ 0))](0) +$
 $(inouts_0\ (n - Suc\ 0)) \bullet [x0 + sum-hd-signal\ inouts_0\ (n - Suc\ (Suc\ 0))](Suc\ 0)$
apply (*simp add: 1 2*)
using $a1\ a2\ 3$
by *simp*
qed

lemma *Solvable-unique-is-solvable*:

assumes *Solvable-unique i1 o1 m n (f)*
shows *Solvable i1 o1 m n (f)*
using *assms* **apply** (*simp add: Solvable-unique-def Solvable-def*)
apply (*clarify*)
by *blast*

unique-solution-integrator: the integrator diagram has a unique solution.

lemma *unique-solution-integrator*:

fixes $inouts_0::nat \Rightarrow real\ list$
assumes $s1: \forall n. length(inouts_0\ n) = 1$
shows $\exists!xx. (\forall n. (n = 0 \longrightarrow xx\ 0 = x0) \wedge$
 $(0 < n \longrightarrow xx\ n = hd((inouts_0\ (n - Suc\ 0))) + xx\ (n - Suc\ 0)))$
apply (*rule ex-ex1I*)
apply (*rule-tac x = $\lambda na. (if\ na = 0\ then\ x0\ else\ (x0 + (\sum i \in \{0..(na-1)\}. hd((inouts_0\ i))))$ in*
 exI)
apply (*simp*)
apply (*rule allI*)
proof –
fix $n::nat$

```

show  $\neg n \leq \text{Suc } 0 \longrightarrow$ 
   $(\sum i = 0..n - \text{Suc } 0. \text{hd } (\text{inouts}_0 i)) =$ 
   $\text{hd } (\text{inouts}_0 (n - \text{Suc } 0)) + (\sum i = 0..n - \text{Suc } (\text{Suc } 0). \text{hd } (\text{inouts}_0 i))$ 
proof (induct n)
  case 0
  thus ?case by auto
next
  case (Suc n) note IH = this
  { assume Suc n = 1
    hence ?case by auto
  }
  also {
    assume Suc n > 1
    {
      assume Suc n = 2
      hence ?case by auto
    }
    also {
      assume Suc n > 2
      have ?case
      by (smt One-nat-def Suc-diff-Suc  $\langle 1 < \text{Suc } n \rangle$  sum.atLeast0-atMost-Suc)
    }
  }
}

ultimately show ?case
by (smt One-nat-def Suc-1 Suc-lessI cancel-comm-monoid-add-class.diff-cancel
  diff-Suc-1 not-less sum.atLeast0-atMost-Suc)
qed
next
fix xx:: nat  $\Rightarrow$  real and y:: nat  $\Rightarrow$  real
assume a1:  $\forall n. (n = 0 \longrightarrow xx\ 0 = x0) \wedge (0 < n \longrightarrow xx\ n = \text{hd } (\text{inouts}_0 (n - \text{Suc } 0)) + xx\ (n - \text{Suc } 0))$ 
assume a2:  $\forall n. (n = 0 \longrightarrow y\ 0 = x0) \wedge (0 < n \longrightarrow y\ n = \text{hd } (\text{inouts}_0 (n - \text{Suc } 0)) + y\ (n - \text{Suc } 0))$ 
have 1:  $\forall n. xx\ n = y\ n$ 
apply (rule allI)
proof -
  fix n::nat
  show xx n = y n
  proof (induct n)
  case 0
  then show ?case
  using a1 a2 by simp
  next
  case (Suc n) note IH = this
  then show ?case
  using a1 a2 by simp
  qed
qed
show xx = y
using 1 fun-eq by (blast)
qed

```

lemma *FBlock-feedback:*

assumes *s1: SimBlock m n (FBlock $(\lambda x n. \text{True})$ m n f)*

assumes $s2$: *Solvable-unique* $i1$ $o1$ m n (f)
shows $(FBlock (\lambda x n. True) m n f) f_D (i1, o1)$
 $= (FBlock (\lambda x n. True) (m-1) (n-1))$
 $(\lambda x na. ((f-PostFD o1) o f o (f-PreFD (Solution i1 o1 m n f x) i1)) x na))$

proof –

have $inps-1$: $inps (FBlock (\lambda x n. True) m n f) = m$
using $s1$ **by** (*simp add: inps-P*)
have $outps-1$: $outps (FBlock (\lambda x n. True) m n f) = n$
using $s1$ **by** (*simp add: outps-P*)
have $i1-lt-m$: $i1 < m$
using $s2$ **by** (*simp add: Solvable-unique-def*)
have $o1-lt-n$: $o1 < n$
using $s2$ **by** (*simp add: Solvable-unique-def*)
have 1 : $(FBlock (\lambda x n. True) m n f) f_D (i1, o1) = (true \vdash_n (\exists x \cdot$
 $(\forall n \cdot \#_u(\$inouts(\langle n \rangle)_a) =_u \langle m - Suc 0 \rangle \wedge$
 $\#_u(\$inouts'(\langle n \rangle)_a) =_u \langle m \rangle \wedge \$inouts'(\langle n \rangle)_a =_u \langle f-PreFD x i1 \rangle (\$inouts)_a(\langle n \rangle)_a)$

;;

$((\forall na \cdot \#_u(\$inouts(\langle na \rangle)_a) =_u \langle m \rangle \wedge$
 $\#_u(\$inouts'(\langle na \rangle)_a) =_u \langle n \rangle \wedge \langle f \rangle (\$inouts)_a(\langle na \rangle)_a =_u \$inouts'(\langle na \rangle)_a) \wedge$
 $(\forall x \cdot \forall na \cdot \#_u(\langle x na \rangle) =_u \langle m \rangle \Rightarrow \#_u(\langle f x na \rangle) =_u \langle n \rangle))$; ;
 $(\forall na \cdot \#_u(\$inouts(\langle na \rangle)_a) =_u \langle n \rangle \wedge$
 $\#_u(\$inouts'(\langle na \rangle)_a) =_u \langle n - Suc 0 \rangle \wedge$
 $\$inouts'(\langle na \rangle)_a =_u \langle f-PostFD o1 \rangle (\$inouts)_a(\langle na \rangle)_a \wedge$
 $\langle uapply \rangle (\$inouts(\langle na \rangle)_a)_a(\langle o1 \rangle)_a =_u \langle x na \rangle))$

apply (*simp add: inps-1 outps-1*)
apply (*simp add: PreFD-def PostFD-def FBlock-def Solution-def*)
apply (*simp add: ndesign-composition-wp wp-upred-def*)
by (*rel-simp*)
have 2 : $(true \vdash_n (\exists x \cdot$
 $(\forall n \cdot \#_u(\$inouts(\langle n \rangle)_a) =_u \langle m - Suc 0 \rangle \wedge$
 $\#_u(\$inouts'(\langle n \rangle)_a) =_u \langle m \rangle \wedge \$inouts'(\langle n \rangle)_a =_u \langle f-PreFD x i1 \rangle (\$inouts)_a(\langle n \rangle)_a)$

;;

$((\forall na \cdot \#_u(\$inouts(\langle na \rangle)_a) =_u \langle m \rangle \wedge$
 $\#_u(\$inouts'(\langle na \rangle)_a) =_u \langle n \rangle \wedge \langle f \rangle (\$inouts)_a(\langle na \rangle)_a =_u \$inouts'(\langle na \rangle)_a) \wedge$
 $(\forall x \cdot \forall na \cdot \#_u(\langle x na \rangle) =_u \langle m \rangle \Rightarrow \#_u(\langle f x na \rangle) =_u \langle n \rangle))$; ;
 $(\forall na \cdot \#_u(\$inouts(\langle na \rangle)_a) =_u \langle n \rangle \wedge$
 $\#_u(\$inouts'(\langle na \rangle)_a) =_u \langle n - Suc 0 \rangle \wedge$
 $\$inouts'(\langle na \rangle)_a =_u \langle f-PostFD o1 \rangle (\$inouts)_a(\langle na \rangle)_a \wedge$
 $\langle uapply \rangle (\$inouts(\langle na \rangle)_a)_a(\langle o1 \rangle)_a =_u \langle x na \rangle))$

$= (FBlock (\lambda x n. True) (m-1) (n-1))$
 $(\lambda x na. ((f-PostFD o1) o f o (f-PreFD (Solution i1 o1 m n f x) i1)) x na))$

apply (*simp add: FBlock-def Solution-def*)
apply (*rule ref-eq*)
apply (*rule ndesign-refine-intro, simp+*)
apply (*rel-simp*)
apply (*rule-tac x = (SOME xx. \forall n. xx n = f (f-PreFD xx i1 inouts_v) n!(o1)) in exI*)
apply (*rule-tac x = \lambda na. f-PreFD (SOME xx. \forall n. xx n = f (f-PreFD xx i1 inouts_v) n!(o1))*
 $i1 inouts_v na$ **in** $exI, simp$)

apply (*rule conjI*)
apply (*simp add: f-PreFD-def*)
using $i1-lt-m$ **apply** *linarith*
apply (*rule-tac x = \lambda na. (f (f-PreFD (SOME xx. \forall n. xx n = f (f-PreFD xx i1 inouts_v) n!(o1))*
 $i1 inouts_v) na)$ **in** $exI, simp$)

apply (*rule conjI*)
apply (*simp add: f-PreFD-def*)


```

apply (rule conjI)
using i1-lt-m apply linarith

defer
apply (rule conjI)
using SimBlock-FBlock-fn s1 apply blast
apply (rule allI, rule conjI)

defer
defer
apply (rule ndesign-refine-intro, simp+)
apply (rel-simp)
apply (rule conjI)
defer
apply (simp add: f-PreFD-def f-PostFD-def)
using o1-lt-n apply linarith
prefer 3
proof -
  fix inouts_v::nat  $\Rightarrow$  real list and inouts_v'::nat  $\Rightarrow$  real list and x::nat
  assume a1:  $\forall x. \text{length}(\text{inouts}_v\ x) = m - \text{Suc}\ 0 \wedge$ 
     $\text{length}(\text{inouts}_v'\ x) = n - \text{Suc}\ 0 \wedge$ 
    f-PostFD o1 (f (f-PreFD (SOME xx.  $\forall n. xx\ n = f$  (f-PreFD xx i1 inouts_v) n!(o1)) i1 inouts_v))
x = inouts_v' x
  let ?P =  $\lambda xx. \forall n. xx\ n = f$  (f-PreFD xx i1 inouts_v) n!(o1)
  have 1: (?P (SOME xx. ?P xx))
    apply (rule someI-ex[of ?P])
    using s2 apply (simp add: Solvable-unique-def)
    using a1 by blast
  show f (f-PreFD (SOME xx. ?P xx) i1 inouts_v) x!(o1) = (SOME xx. ?P xx) x
    by (simp add: 1)
next
  fix inouts_v inouts_v'
  assume a1:  $\forall x. \text{length}(\text{inouts}_v\ x) = m - \text{Suc}\ 0 \wedge$ 
     $\text{length}(\text{inouts}_v'\ x) = n - \text{Suc}\ 0 \wedge$ 
    f-PostFD o1 (f (f-PreFD (SOME xx.  $\forall n. xx\ n = f$  (f-PreFD xx i1 inouts_v) n!(o1)) i1 inouts_v))
x =
  inouts_v' x
  assume a2:  $\forall x\ xa. \text{length}(x\ xa) = m - \text{Suc}\ 0 \longrightarrow$ 
     $\text{length}(f\text{-PostFD}\ o1\ (f\ (f\text{-PreFD}\ (SOME\ xx.\ \forall n.\ xx\ n = f\ (f\text{-PreFD}\ xx\ i1\ x)\ n!(o1))\ i1\ x)))\ i1\ xa) =$ 
xa) =
  n - Suc 0
  from a1 have a1':  $\forall x. \text{length}(\text{inouts}_v\ x) = m - \text{Suc}\ 0$ 
    by (simp)
  have  $\forall na. \text{length}((f\text{-PreFD}\ (SOME\ xx.\ \forall n.\ xx\ n = f\ (f\text{-PreFD}\ xx\ i1\ \text{inouts}_v)\ n!(o1))\ i1\ \text{inouts}_v))$ 
na) = m
    using a1' f-PreFD-def apply (simp)
    using i1-lt-m by linarith
  then show  $\forall x. \text{length}(f\ (f\text{-PreFD}\ (SOME\ xx.\ \forall n.\ xx\ n = f\ (f\text{-PreFD}\ xx\ i1\ \text{inouts}_v)\ n!(o1))\ i1\ \text{inouts}_v)\ x) = n$ 
    using SimBlock-FBlock-fn s1 by blast
next
  fix inouts_v inouts_v' x
  assume a1:  $\forall x. \text{length}(\text{inouts}_v\ x) = m - \text{Suc}\ 0 \wedge$ 
     $\text{length}(\text{inouts}_v'\ x) = n - \text{Suc}\ 0 \wedge$ 
    f-PostFD o1 (f (f-PreFD (SOME xx.  $\forall n. xx\ n = f$  (f-PreFD xx i1 inouts_v) n!(o1)) i1 inouts_v))

```

```

x =
  inouts_v' x
  assume a2:  $\forall x xa. \text{length}(x xa) = m - \text{Suc } 0 \longrightarrow$ 
     $\text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 x) n!(o1)) i1 x))$ 
xa) =
  n - Suc 0
  from a1 have a1':  $\forall x. \text{length}(\text{inouts}_v x) = m - \text{Suc } 0$ 
  by (simp)
  have  $\forall na. \text{length}((f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v) n!(o1)) i1 \text{inouts}_v)$ 
na) = m
  using a1' f-PreFD-def apply (simp)
  using i1-lt-m by linarith
  then show  $\text{length}(f (f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v) n!(o1)) i1$ 
inouts_v) x) = n
  using SimBlock-FBlock-fn s1 by blast
next
fix inouts_v::nat  $\Rightarrow$  real list and inouts_v':nat  $\Rightarrow$  real list and x::nat  $\Rightarrow$  real and
  inouts_v'':nat  $\Rightarrow$  real list and inouts_v''':nat  $\Rightarrow$  real list
assume a1:  $\forall xa. \text{length}(\text{inouts}_v xa) = m - \text{Suc } 0 \wedge \text{inouts}_v'' xa = f\text{-PreFD } x i1 \text{inouts}_v xa$ 
assume a2:  $\forall xa. \text{length}(f\text{-PreFD } x i1 \text{inouts}_v xa) = m \wedge f \text{inouts}_v'' xa = \text{inouts}_v''' xa$ 
assume a3:  $\forall xa. \text{length}(\text{inouts}_v''' xa) = n \wedge \text{length}(\text{inouts}_v' xa) = n - \text{Suc } 0 \wedge$ 
   $\text{inouts}_v' xa = f\text{-PostFD } o1 \text{inouts}_v''' xa \wedge \text{inouts}_v''' xa!(o1) = x xa$ 
have unique-sol:
  ( $\exists! (xx::nat \Rightarrow \text{real}).$ 
  ( $\forall n. (xx n = (f (\lambda n1. f\text{-PreFD } xx i1 \text{inouts}_v n1) n)!(o1))$ ))
  using s2 a1 by (simp add: Solvable-unique-def)
from a1 a2 have  $\forall xa. \text{inouts}_v''' xa = f \text{inouts}_v'' xa$ 
  by simp
then have  $\forall xa. \text{inouts}_v''' xa = f (f\text{-PreFD } x i1 \text{inouts}_v) xa$ 
  using a1 by presburger
then have 0:  $\text{inouts}_v''' = f (f\text{-PreFD } x i1 \text{inouts}_v)$ 
  by (rule fun-eq)
have 1:  $(\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v) n!(o1)) = x$ 
  apply (rule some-equality)
  using 0 a3 unique-sol by auto
then have 2:  $\forall n. f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v)$ 
n!(o1)) i1 inouts_v)) n
  =  $f\text{-PostFD } o1 (f (f\text{-PreFD } x i1 \text{inouts}_v)) n$ 
  by blast
then have 3:  $\forall n. f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v)$ 
n!(o1)) i1 inouts_v)) n
  =  $f\text{-PostFD } o1 \text{inouts}_v''' n$ 
  using 0 by blast
show  $\forall x. \text{length}(f\text{-PostFD } o1 \text{inouts}_v''' x) = n - \text{Suc } 0 \wedge$ 
   $f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{SOME } xx. \forall n. xx n = f (f\text{-PreFD } xx i1 \text{inouts}_v) n!(o1)) i1 \text{inouts}_v))$ 
x
  =  $f\text{-PostFD } o1 \text{inouts}_v''' x$ 
  apply (rule allI, rule conjI)
  apply (simp add: f-PostFD-def)
  using a3 o1-lt-n apply auto[1]
  using 3 by blast
qed
show ?thesis
  using 1 by (simp add: 2)
qed

```

lemma *unique-solution*:

assumes $s1$: *Solvable-unique* $i1$ $o1$ m n (f)
assumes $s2$: *is-Solution* $i1$ $o1$ m n (f) (xx)
assumes $s3$: $\forall n. \text{length}(\text{ins } n) = m - 1$
shows $xx \text{ ins} = (\text{Solution } i1 \ o1 \ m \ n \ f \ \text{ins})$
using $s1$ $s2$ **apply** (*simp add: Solution-def Solvable-unique-def is-Solution-def*)
apply (*clarify*)
proof –
assume $a1$: $\forall \text{inouts}_0. (\forall x. \text{length}(\text{inouts}_0 \ x) = m - \text{Suc } 0) \longrightarrow$
 $(\forall n. xx \ \text{inouts}_0 \ n = f \ (f\text{-PreFD } (xx \ \text{inouts}_0) \ i1 \ \text{inouts}_0) \ n!(o1))$
assume $a2$: $\forall \text{inouts}_0. (\forall x. \text{length}(\text{inouts}_0 \ x) = m - \text{Suc } 0) \longrightarrow$
 $(\exists!xx. \forall n. xx \ n = f \ (f\text{-PreFD } xx \ i1 \ \text{inouts}_0) \ n!(o1))$
have (*SOME* $xx. \forall n. xx \ n = f \ (f\text{-PreFD } xx \ i1 \ \text{ins}) \ n!(o1)$) = $xx \ \text{ins}$
apply (*rule some-equality*)
using $a1$ $s3$ **apply** *simp*
using $a2$ **apply** (*simp add: Ex1-def*)
proof –
fix xxa
assume $a3$: $\forall n. xxa \ n = f \ (f\text{-PreFD } xxa \ i1 \ \text{ins}) \ n!(o1)$
assume $a4$: $\forall \text{inouts}_0.$
 $(\forall x. \text{length}(\text{inouts}_0 \ x) = m - \text{Suc } 0) \longrightarrow$
 $(\exists x. (\forall n. x \ n = f \ (f\text{-PreFD } x \ i1 \ \text{inouts}_0) \ n!(o1)) \wedge$
 $(\forall y. (\forall n. y \ n = f \ (f\text{-PreFD } y \ i1 \ \text{inouts}_0) \ n!(o1)) \longrightarrow y = x))$
from $a4$ $s3$ **have** 1 : $(\exists x. (\forall n. x \ n = f \ (f\text{-PreFD } x \ i1 \ \text{ins}) \ n!(o1)) \wedge$
 $(\forall y. (\forall n. y \ n = f \ (f\text{-PreFD } y \ i1 \ \text{ins}) \ n!(o1)) \longrightarrow y = x))$
by *simp*
from $s2$ **have** 2 : $\forall n. (xx \ \text{ins}) \ n = f \ (f\text{-PreFD } (xx \ \text{ins}) \ i1 \ \text{ins}) \ n!(o1)$
using $a1$ $s3$ **by** *simp*
show $xxa = xx \ \text{ins}$
using $a3$ $a4$ $s3$ 1 2 **by** *blast*
qed
then show $xx \ \text{ins} = (\text{SOME } xx. \forall n. xx \ n = f \ (f\text{-PreFD } xx \ i1 \ \text{ins}) \ n!(o1))$
by *simp*
qed

lemma *FBlock-feedback'*:

assumes $s1$: *SimBlock* m n (*FBlock* $(\lambda x \ n. \text{True}) \ m \ n \ f$)
assumes $s2$: *Solvable-unique* $i1$ $o1$ m n (f)
assumes $s3$: *is-Solution* $i1$ $o1$ m n (f) (xx)
shows (*FBlock* $(\lambda x \ n. \text{True}) \ m \ n \ f$) $f_D \ (i1, \ o1)$
 $= (\text{FBlock } (\lambda x \ n. \text{True}) \ (m - 1) \ (n - 1)$
 $(\lambda x \ na. ((f\text{-PostFD } o1) \ o \ f \ o \ (f\text{-PreFD } (xx \ x) \ i1)) \ x \ na))$
using $s1$ $s2$ *FBlock-feedback* **apply** (*simp*)
proof –
have $i1\text{-lt-}m$: $i1 < m$
using $s2$ **by** (*simp add: Solvable-unique-def*)
have $o1\text{-lt-}n$: $o1 < n$
using $s2$ **by** (*simp add: Solvable-unique-def*)
show *FBlock* $(\lambda x \ n. \text{True}) \ (m - \text{Suc } 0) \ (n - \text{Suc } 0)$
 $(\lambda x. f\text{-PostFD } o1 \ (f \ (f\text{-PreFD } (\text{Solution } i1 \ o1 \ m \ n \ f \ x) \ i1 \ x))) =$
 $\text{FBlock } (\lambda x \ n. \text{True}) \ (m - \text{Suc } 0) \ (n - \text{Suc } 0) \ (\lambda x. f\text{-PostFD } o1 \ (f \ (f\text{-PreFD } (xx \ x) \ i1 \ x)))$
apply (*simp (no-asm) add: FBlock-def*)
apply (*rel-simp*)
apply (*rule iffI, clarify*)

```

defer
apply (clarify)
defer
proof -
  fix  $ok_v$   $inouts_v$   $ok_v'$   $inouts_v'$ 
  assume  $a1: \forall x. \text{length}(inouts_v x) = m - \text{Suc } 0 \wedge$ 
     $\text{length}(inouts_v' x) = n - \text{Suc } 0 \wedge$ 
     $f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{Solution } i1 o1 m n f inouts_v) i1 inouts_v)) x = inouts_v' x$ 
  assume  $a2: \forall x xa. \text{length}(x xa) = m - \text{Suc } 0 \longrightarrow$ 
     $\text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{Solution } i1 o1 m n f x) i1 x)) xa) = n - \text{Suc } 0$ 
  have 1:  $\forall x. \text{length}(inouts_v x) = m - \text{Suc } 0$ 
  using  $a1$  by simp
  have 2:  $xx inouts_v = (\text{Solution } i1 o1 m n f inouts_v)$ 
  apply (rule unique-solution)
  using  $s2$  apply (simp)
  using  $s3$  apply (simp)
  using 1 by (simp)
  show  $(\forall x. \text{length}(inouts_v x) = m - \text{Suc } 0 \wedge \text{length}(inouts_v' x) = n - \text{Suc } 0 \wedge$ 
     $f\text{-PostFD } o1 (f (f\text{-PreFD } (xx inouts_v) i1 inouts_v)) x = inouts_v' x) \wedge$ 
     $(\forall x xa. \text{length}(x xa) = m - \text{Suc } 0 \longrightarrow \text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (xx x) i1 x)) xa) =$ 
 $n - \text{Suc } 0)$ 
  apply (rule conjI)
  using 2  $a1$  apply simp
  apply (rule allI)
  apply (clarify)
  proof -
    fix  $x::nat \Rightarrow$  real list and  $xa::nat$ 
    assume  $a11: \text{length}(x xa) = m - \text{Suc } 0$ 
    have 1:  $\text{length}((f\text{-PreFD } (xx x) i1 x) xa) = m$ 
    using  $a11$  apply (simp add: f-PreFD-def)
    using  $i1\text{-lt-}m$  by linarith
    have 2:  $\text{length}((f (f\text{-PreFD } (xx x) i1 x)) xa) = n$ 
    using 1  $\text{SimBlock-FBlock-fn } s1$  by blast
    show  $\text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (xx x) i1 x)) xa) = n - \text{Suc } 0$ 
    apply (simp add: f-PostFD-def f-PreFD-def)
    using 1 2  $o1\text{-lt-}n$  by linarith
  qed
next
  fix  $ok_v$   $inouts_v$   $ok_v'$   $inouts_v'$ 
  assume  $a1: \forall x. \text{length}(inouts_v x) = m - \text{Suc } 0 \wedge \text{length}(inouts_v' x) = n - \text{Suc } 0 \wedge$ 
     $f\text{-PostFD } o1 (f (f\text{-PreFD } (xx inouts_v) i1 inouts_v)) x = inouts_v' x$ 
  assume  $a2: \forall x xa. \text{length}(x xa) = m - \text{Suc } 0 \longrightarrow \text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (xx x) i1$ 
 $x)) xa) = n - \text{Suc } 0$ 
  have 1:  $\forall x. \text{length}(inouts_v x) = m - \text{Suc } 0$ 
  using  $a1$  by simp
  have 2:  $xx inouts_v = (\text{Solution } i1 o1 m n f inouts_v)$ 
  apply (rule unique-solution)
  using  $s2$  apply (simp)
  using  $s3$  apply (simp)
  using 1 by (simp)
  show  $(\forall x. \text{length}(inouts_v x) = m - \text{Suc } 0 \wedge \text{length}(inouts_v' x) = n - \text{Suc } 0 \wedge$ 
     $f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{Solution } i1 o1 m n f inouts_v) i1 inouts_v)) x = inouts_v' x) \wedge$ 
     $(\forall x xa. \text{length}(x xa) = m - \text{Suc } 0 \longrightarrow$ 
     $\text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (\text{Solution } i1 o1 m n f x) i1 x)) xa) = n - \text{Suc } 0)$ 
  apply (rule conjI)

```

```

using 2 a1 apply auto[1]
apply (rule allI)
apply (clarify)
proof -
  fix x::nat  $\Rightarrow$  real list and xa::nat
  assume a11: length (x xa) = m - Suc 0
  have 1: length((f-PreFD (Solution i1 o1 m n f x) i1 x) xa) = m
    using a11 apply (simp add: f-PreFD-def)
    using i1-lt-m by linarith
  have 2: length((f (f-PreFD (Solution i1 o1 m n f x) i1 x)) xa) = n
    using 1 SimBlock-FBlock-fn s1 by blast
  show length(f-PostFD o1 (f (f-PreFD (Solution i1 o1 m n f x) i1 x)) xa) = n - Suc 0
    apply (simp add: f-PostFD-def f-PreFD-def)
    using 1 2 o1-lt-n by linarith
qed
qed
qed

```

lemma FBlock-feedback-ref:

```

assumes s1: SimBlock m n (FBlock ( $\lambda$ x n. True) m n f)
assumes s2: Solvable i1 o1 m n (f)
shows (FBlock ( $\lambda$ x n. True) m n f) fD (i1, o1)
   $\sqsubseteq$  (FBlock ( $\lambda$ x n. True) (m-1) (n-1)
    ( $\lambda$ x na. ((f-PostFD o1) o f o (f-PreFD (Solution i1 o1 m n f x) i1)) x na))
proof -
  have inps-1: inps (FBlock ( $\lambda$ x n. True) m n f) = m
    using s1 by (simp add: inps-P)
  have outps-1: outps (FBlock ( $\lambda$ x n. True) m n f) = n
    using s1 by (simp add: outps-P)
  have i1-lt-m: i1 < m
    using s2 by (simp add: Solvable-def)
  have o1-lt-n: o1 < n
    using s2 by (simp add: Solvable-def)
  have 1: (FBlock ( $\lambda$ x n. True) m n f) fD (i1, o1) = (true  $\vdash_n$  ( $\exists$  x  $\cdot$ 
    ( $\forall$  n  $\cdot$  #u( $\$$ inouts( $\langle\langle n \rangle\rangle$ )a) =u  $\langle\langle m - Suc 0 \rangle\rangle$   $\wedge$ 
      #u( $\$$ inouts'( $\langle\langle n \rangle\rangle$ )a) =u  $\langle\langle m \rangle\rangle$   $\wedge$   $\$$ inouts'( $\langle\langle n \rangle\rangle$ )a =u  $\langle\langle f\text{-PreFD } x \text{ } i1 \rangle\rangle$ ( $\$$ inouts)a( $\langle\langle n \rangle\rangle$ )a
    ))
  ;;
  (( $\forall$  na  $\cdot$  #u( $\$$ inouts( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle m \rangle\rangle$   $\wedge$ 
    #u( $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle n \rangle\rangle$   $\wedge$   $\langle\langle f \rangle\rangle$ ( $\$$ inouts)a( $\langle\langle na \rangle\rangle$ )a =u  $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a  $\wedge$ 
    ( $\forall$  x  $\cdot$   $\forall$  na  $\cdot$  #u( $\langle\langle x \text{ } na \rangle\rangle$ ) =u  $\langle\langle m \rangle\rangle$   $\Rightarrow$  #u( $\langle\langle f \text{ } x \text{ } na \rangle\rangle$ ) =u  $\langle\langle n \rangle\rangle$ )) ;;
  ( $\forall$  na  $\cdot$  #u( $\$$ inouts( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle n \rangle\rangle$   $\wedge$ 
    #u( $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle n - Suc 0 \rangle\rangle$   $\wedge$ 
     $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a =u  $\langle\langle f\text{-PostFD } o1 \rangle\rangle$ ( $\$$ inouts)a( $\langle\langle na \rangle\rangle$ )a  $\wedge$ 
     $\langle\langle u\text{apply} \rangle\rangle$ ( $\$$ inouts( $\langle\langle na \rangle\rangle$ )a)a( $\langle\langle o1 \rangle\rangle$ )a =u  $\langle\langle x \text{ } na \rangle\rangle$ ))
  apply (simp add: inps-1 outps-1)
  apply (simp add: PreFD-def PostFD-def FBlock-def Solution-def)
  apply (simp add: ndesign-composition-wp wp-upred-def)
  by (rel-simp)
  have 2: (true  $\vdash_n$  ( $\exists$  x  $\cdot$ 
    ( $\forall$  n  $\cdot$  #u( $\$$ inouts( $\langle\langle n \rangle\rangle$ )a) =u  $\langle\langle m - Suc 0 \rangle\rangle$   $\wedge$ 
      #u( $\$$ inouts'( $\langle\langle n \rangle\rangle$ )a) =u  $\langle\langle m \rangle\rangle$   $\wedge$   $\$$ inouts'( $\langle\langle n \rangle\rangle$ )a =u  $\langle\langle f\text{-PreFD } x \text{ } i1 \rangle\rangle$ ( $\$$ inouts)a( $\langle\langle n \rangle\rangle$ )a
    ))
  ;;
  (( $\forall$  na  $\cdot$  #u( $\$$ inouts( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle m \rangle\rangle$   $\wedge$ 
    #u( $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a) =u  $\langle\langle n \rangle\rangle$   $\wedge$   $\langle\langle f \rangle\rangle$ ( $\$$ inouts)a( $\langle\langle na \rangle\rangle$ )a =u  $\$$ inouts'( $\langle\langle na \rangle\rangle$ )a  $\wedge$ 
    ( $\forall$  x  $\cdot$   $\forall$  na  $\cdot$  #u( $\langle\langle x \text{ } na \rangle\rangle$ ) =u  $\langle\langle m \rangle\rangle$   $\Rightarrow$  #u( $\langle\langle f \text{ } x \text{ } na \rangle\rangle$ ) =u  $\langle\langle n \rangle\rangle$ )) ;;

```

```

(∀ na · #u($inouts(«na»)a) =u «n» ∧
  #u($inouts'(«na»)a) =u «n - Suc 0» ∧
  $inouts'(«na»)a =u «f-PostFD o1»($inouts)a(«na»)a ∧
  «uapply»($inouts(«na»)a)(«o1»)a =u «x na»)))
⊆ (FBlock (λx n. True) (m-1) (n-1)
  (λx na. ((f-PostFD o1) o f o (f-PreFD (Solution i1 o1 m n f x) i1)) x na))
apply (simp add: FBlock-def Solution-def)
apply (rule ndesign-refine-intro, simp+)
apply (rel-simp)
apply (rule-tac x = (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1)) in exI)
apply (rule-tac x = λna. f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1))
  i1 inoutsv na in exI, simp)

apply (rule conjI)
apply (simp add: f-PreFD-def)
using i1-lt-m apply linarith
apply (rule-tac x = λna. (f (f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1))
  i1 inoutsv) na) in exI, simp)

apply (rule conjI)
apply (simp add: f-PreFD-def)
apply (rule conjI)
using i1-lt-m apply linarith

defer
apply (rule conjI)
using SimBlock-FBlock-fn s1 apply blast
apply (rule allI, rule conjI)

defer
proof -
  fix inoutsv::nat ⇒ real list and inoutsv'::nat ⇒ real list and x::nat
  assume a1: ∀ x. length(inoutsv x) = m - Suc 0 ∧
    length(inoutsv' x) = n - Suc 0 ∧
    f-PostFD o1 (f (f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1)) i1 inoutsv))
x = inoutsv' x
  let ?P = λxx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1)
  have 1: (?P (SOME xx. ?P xx))
    apply (rule someI-ex[of ?P])
    using s2 apply (simp add: Solvable-def)
    using a1 by blast
  show f (f-PreFD (SOME xx. ?P xx) i1 inoutsv) x!(o1) = (SOME xx. ?P xx) x
    by (simp add: 1)
next
  fix inoutsv inoutsv'
  assume a1: ∀ x. length(inoutsv x) = m - Suc 0 ∧
    length(inoutsv' x) = n - Suc 0 ∧
    f-PostFD o1 (f (f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1)) i1 inoutsv))
x =
  inoutsv' x
  assume a2: ∀ x xa. length(x xa) = m - Suc 0 →
    length(f-PostFD o1 (f (f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 x) n!(o1)) i1 x))
xa) =
  n - Suc 0
  from a1 have a1': ∀ x. length(inoutsv x) = m - Suc 0
    by (simp)
  have ∀ na. length((f-PreFD (SOME xx. ∀ n. xx n = f (f-PreFD xx i1 inoutsv) n!(o1)) i1 inoutsv)

```

```

na) = m
  using a1' f-PreFD-def apply (simp)
  using i1-lt-m by linarith
  then show  $\forall x. \text{length}(f (f\text{-PreFD } (SOME\ xx. \forall n. xx\ n = f (f\text{-PreFD } xx\ i1\ inouts_v)\ n!(o1))\ i1\ inouts_v)\ x) = n$ 
    using SimBlock-FBlock-fn s1 by blast
  next
  fix inouts_v inouts_v' x
  assume a1:  $\forall x. \text{length}(inouts_v\ x) = m - Suc\ 0 \wedge$ 
     $\text{length}(inouts_v'\ x) = n - Suc\ 0 \wedge$ 
     $f\text{-PostFD } o1 (f (f\text{-PreFD } (SOME\ xx. \forall n. xx\ n = f (f\text{-PreFD } xx\ i1\ inouts_v)\ n!(o1))\ i1\ inouts_v))$ 
x =
  inouts_v' x
  assume a2:  $\forall x\ xa. \text{length}(x\ xa) = m - Suc\ 0 \longrightarrow$ 
     $\text{length}(f\text{-PostFD } o1 (f (f\text{-PreFD } (SOME\ xx. \forall n. xx\ n = f (f\text{-PreFD } xx\ i1\ x)\ n!(o1))\ i1\ x))$ 
xa) =
  n - Suc 0
  from a1 have a1':  $\forall x. \text{length}(inouts_v\ x) = m - Suc\ 0$ 
  by (simp)
  have  $\forall na. \text{length}((f\text{-PreFD } (SOME\ xx. \forall n. xx\ n = f (f\text{-PreFD } xx\ i1\ inouts_v)\ n!(o1))\ i1\ inouts_v)$ 
na) = m
  using a1' f-PreFD-def apply (simp)
  using i1-lt-m by linarith
  then show  $\text{length}(f (f\text{-PreFD } (SOME\ xx. \forall n. xx\ n = f (f\text{-PreFD } xx\ i1\ inouts_v)\ n!(o1))\ i1\ inouts_v)\ x) = n$ 
    using SimBlock-FBlock-fn s1 by blast
  qed
  show ?thesis
  by (metis 1 2)
  qed

```

lemma *SimBlock-FBlock-feedback* [*simblock-healthy*]:

assumes *s1*: *SimBlock* *m n* (*FBlock* ($\lambda x\ n. True$) *m n f*)

assumes *s2*: *Solvable* *i1 o1 m n* (*f*)

shows *SimBlock* (*m-1*) (*n-1*) ((*FBlock* ($\lambda x\ n. True$) *m n f*) *f_D* (*i1*, *o1*))

proof –

have *m1-ge-0*: $(m - (Suc\ 0)) \geq 0$

using *s2* **by** (*simp add: Solvable-def*)

have *m1-gt-0*: $m > 0$

using *s2* **by** (*simp add: Solvable-def*)

have *inps-1*: *inps* (*FBlock* ($\lambda x\ n. True$) *m n f*) = *m*

using *inps-outps s1* **by** *blast*

have *outps-1*: *outps* (*FBlock* ($\lambda x\ n. True$) *m n f*) = *n*

using *inps-outps s1* **by** *blast*

have *i1-le-m*: $i1 \leq m - Suc\ 0$

using *s2* **apply** (*simp add: Solvable-def*)

by *linarith*

have *o1-le-n*: $o1 \leq n - Suc\ 0$

using *s2* **apply** (*simp add: Solvable-def*)

by *linarith*

obtain *inouts₀*: $\text{nat} \Rightarrow \text{real list}$ **where** *P0*: $\forall x. \text{length}(inouts_0\ x) = (m - 1)$

using *m1-gt-0 list-len-avail*

by *blast*

have $(\forall inouts_0. (\forall x. \text{length}(inouts_0\ x) = (m - 1))$

$\longrightarrow (\exists xx.$

```

    (∀ n. (xx n =
      (f (λn1.
        ((take i1 (inouts0 n1))•(xx n1)#(drop i1 (inouts0 n1)))
        ) n)!o1
      )))
  using s2 by (simp add: Solvable-def f-PreFD-def)
  then have 1: ∃ xx. (∀ n. (xx n = (f (λn1. ((take i1 (inouts0 n1))•(xx n1)#(drop i1 (inouts0 n1))))
n)!o1))
  apply (simp)
  using P0 by simp
  obtain xx::nat ⇒ real
  where P1: (∀ n. (xx n = (f (λn1. ((take i1 (inouts0 n1))•(xx n1)#(drop i1 (inouts0 n1))))
n)!o1
  ))
  using 1 P0 by blast
  have 2: Suc (m - Suc 0) = m
  using m1-gt-0 by simp
  show ?thesis
  apply (simp add: SimBlock-def inps-1 outps-1 PreFD-def PostFD-def)
  apply (simp add: FBlock-def)
  apply (rel-auto)
  apply (simp add: f-blocks)

  apply (rule-tac x = inouts0 in exI)
  apply (rule-tac x = λna.
    (remove-at (f (λn1. ((take i1 (inouts0 n1))•[xx n1]•(drop i1 (inouts0 n1))))
na) o1) in exI)
  apply (rule-tac x = xx in exI)
  apply (rule-tac x = True in exI, simp)
  apply (rule-tac x = λna. (
    (λn1. ((take i1 (inouts0 n1))•[xx n1]•(drop i1 (inouts0 n1))))
na) in exI)
  apply (simp)
  apply (rule conjI)
  apply (rule allI)
  apply (rule conjI)
  using P0 apply (simp)
  apply (simp add: 2 P0)
  apply (rule-tac x = True in exI, simp)
  apply (rule-tac x = λna.
    ((f (λn1. ((take i1 (inouts0 n1))•[xx n1]•(drop i1 (inouts0 n1))))
na)) in exI)
  apply (simp)
  apply (rule conjI)
  using 2 P0 SimBlock-FBlock-fn s1
  apply (smt One-nat-def add-Suc-right append-take-drop-id length-Cons length-append)
  apply (rule conjI)
  using SimBlock-FBlock-fn s1 apply blast
  apply (rule allI)
  apply (rule conjI)
  using SimBlock-FBlock-fn s1
  apply (smt 2 One-nat-def P0 add-Suc-right append-take-drop-id length-Cons length-append)
  apply (rule conjI)
  defer
  using P1 apply metis
  proof -
    fix x
    have 1: length(f (λn1. take i1 (inouts0 n1) • xx n1 # drop i1 (inouts0 n1)) x) = n

```



```

    using 2 P0 SimBlock-FBlock-fn s1
    by (smt One-nat-def add-Suc-right append-take-drop-id length-Cons length-append)
  show min (length(f (λn1. take i1 (inouts0 n1) • xx n1 # drop i1 (inouts0 n1)) x)) o1 +
    (length(f (λn1. take i1 (inouts0 n1) • xx n1 # drop i1 (inouts0 n1)) x) - Suc o1) =
      n - Suc 0
    apply (simp add: 1)
    using o1-le-n by linarith
  qed
qed

```

B.4.5 Split

lemma *SimBlock-Split2* [*simblock-healthy*]:
SimBlock 1 2 (Split2)
 apply (simp add: f-sim-blocks)
 apply (rule *SimBlock-FBlock*)
 apply (simp add: f-blocks)
 apply (rule-tac x = λna. [1] in exI)
 apply force
 by (simp add: f-blocks)

B.5 Blocks

B.5.1 Source

B.5.1.1 Const lemma *SimBlock-Const* [*simblock-healthy*]:
SimBlock 0 1 (Const c0)
 apply (simp add: f-sim-blocks)
 apply (rule *SimBlock-FBlock*)
 apply (simp add: f-blocks)
 apply (rule-tac x = λna. [] in exI)
 apply force
 by (simp add: f-blocks)

B.5.1.2 Pulse Generator

B.5.2 Unit Delay

lemma *SimBlock-UnitDelay* [*simblock-healthy*]:
SimBlock 1 1 (UnitDelay x0)
 apply (simp add: f-sim-blocks)
 apply (rule *SimBlock-FBlock*)
 apply (simp add: f-blocks)
 apply (rule-tac x = λna. [1] in exI)
 apply (rule-tac x = λna. [if na = 0 then x0 else 1] in exI)
 apply (simp)
 by (simp add: f-blocks)

B.5.3 Discrete-Time Integrator

B.5.4 Sum

lemma *SimBlock-Sum2* [*simblock-healthy*]:
SimBlock 2 1 (Sum2)
 apply (simp add: f-sim-blocks)
 apply (rule *SimBlock-FBlock*)
 apply (simp add: f-blocks)

apply (*rule-tac* $x = \lambda na. [1,1]$ **in** exI)
apply (*rule-tac* $x = \lambda na. [2]$ **in** exI)
apply (*simp*)
by (*simp add: f-blocks*)

B.5.5 Product

lemma *SimBlock-Mul2* [*simblock-healthy*]:
SimBlock 2 1 (Mul2)
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*simp add: f-blocks*)
apply (*rule-tac* $x = \lambda na. [1,1]$ **in** exI)
apply (*rule-tac* $x = \lambda na. [1]$ **in** exI)
apply (*simp*)
by (*simp add: f-blocks*)

lemma *SimBlock-Div2* [*simblock-healthy*]:
SimBlock 2 1 (Div2)
apply (*simp add: f-sim-blocks*)
apply (*simp add: SimBlock-def FBlock-def*)
apply (*rel-auto*)
apply (*rule-tac* $x = \lambda na. [1,1]$ **in** exI)
apply (*simp*)
apply (*rule conjI*)
apply (*rule-tac* $x = \lambda na. [1]$ **in** exI)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

B.5.6 Gain

lemma *SimBlock-Gain* [*simblock-healthy*]:
SimBlock 1 1 (Gain k)
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*simp add: f-blocks*)
apply (*rule-tac* $x = \lambda na. [1]$ **in** exI)
apply (*rule-tac* $x = \lambda na. [k]$ **in** exI)
apply (*simp*)
by (*simp add: f-blocks*)

B.5.7 Saturation

lemma *SimBlock-Limit* [*simblock-healthy*]:
assumes $ymin \leq ymax$
shows *SimBlock 1 1 (Limit ymin ymax)*
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*simp add: f-blocks*)
apply (*rule-tac* $x = \lambda na. [ymin]$ **in** exI)
apply (*rule-tac* $x = \lambda na. [ymin]$ **in** exI)
using *assms* **apply** (*simp*)
by (*simp add: f-blocks*)

B.5.8 MinMax

lemma *SimBlock-Min2* [*simblock-healthy*]:
 shows *SimBlock* 2 1 (*Min2*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*simp add: f-blocks*)
 apply (*rule-tac x = λna. [1,2] in exI*)
 apply (*rule-tac x = λna. [1] in exI*)
 apply (*simp*)
 by (*simp add: f-blocks*)

lemma *SimBlock-Max2* [*simblock-healthy*]:
 shows *SimBlock* 2 1 (*Max2*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*simp add: f-blocks*)
 apply (*rule-tac x = λna. [1,2] in exI*)
 apply (*rule-tac x = λna. [2] in exI*)
 apply (*simp*)
 by (*simp add: f-blocks*)

B.5.9 Rounding

lemma *SimBlock-RoundFloor* [*simblock-healthy*]:
 shows *SimBlock* 1 1 (*RoundFloor*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*simp add: f-blocks*)
 apply (*rule-tac x = λna. [1] in exI*)
 apply (*rule-tac x = λna. [1] in exI*)
 apply *auto*[1]
 by (*simp add: f-blocks*)

lemma *SimBlock-RoundCeil* [*simblock-healthy*]:
 shows *SimBlock* 1 1 (*RoundCeil*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*simp add: f-blocks*)
 apply (*rule-tac x = λna. [1] in exI*)
 apply (*rule-tac x = λna. [1] in exI*)
 apply *auto*[1]
 by (*simp add: f-blocks*)

B.5.10 Combinatorial Logic

B.5.11 Logic Operators

B.5.11.1 AND lemma *LAnd* [1,1] = *True*
 by *auto*

lemma *LAnd* [1,1,0] = *False*
 by *auto*

lemma *LAnd-and-not*: *LAnd* [a,b] = (*a* ≠ 0 ∧ *b* ≠ 0)
 by (*simp*)

lemma *LAnd-not-or*: $LAnd [a,b] = (\neg (a = 0 \vee b = 0))$
 by (*simp*)

lemma *SimBlock-LopAND* [*simblock-healthy*]:

assumes $s1: m > 0$
shows *SimBlock* m 1 (*LopAND* m)
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
proof –
obtain $inouts_v :: nat \Rightarrow real\ list$
where $P: \forall na. length(inouts_v\ na) = m \wedge (\forall x < m. ((inouts_v\ na)!x = 0))$
using *list-len-avail'* **by** *fastforce*
have 1: $(\forall x < m. ((inouts_v\ na)!x = 0))$
using P **by** *blast*
have 2: $length(inouts_v\ na) = m$
using P **by** *blast*
from 1 2 **have** 3: (*LAnd* ($inouts_v\ x$) = *False*)
using P $s1$ **by** (*metis LAnd.simps(2) hd-Cons-tl length-0-conv neq0-conv nth-Cons-0*)
show $\exists inouts_v\ inouts_v'$.
 $\forall x. length(inouts_v'\ x) = Suc\ 0 \wedge length(inouts_v\ x) = m \wedge f\text{-LopAND}\ inouts_v\ x = inouts_v'\ x$
apply (*rule-tac* $x = inouts_v$ **in** *exI*)
apply (*simp add: f-blocks*)
apply (*rule-tac* $x = \lambda na. [0]$ **in** *exI*)
using P 3
by (*metis (full-types) LAnd.simps(2) hd-Cons-tl length-0-conv length-Cons nth-Cons-0 s1*)
next
show $\forall x\ na. length(x\ na) = m \longrightarrow length(f\text{-LopAND}\ x\ na) = Suc\ 0$
by (*simp add: f-blocks*)
qed

B.5.11.2 OR lemma *LOr* $[0,0] = False$
 by *auto*

lemma *LOr* $[0,1,0] = True$
 by *auto*

lemma *SimBlock-LopOR* [*simblock-healthy*]:

assumes $s1: m > 0$
shows *SimBlock* m 1 (*LopOR* m)
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
proof –
obtain $inouts_v :: nat \Rightarrow real\ list$
where $P: \forall na. length(inouts_v\ na) = m \wedge (\forall x < m. ((inouts_v\ na)!x = 1))$
using *list-len-avail'* **by** *fastforce*
have 1: $(\forall x < m. ((inouts_v\ na)!x = 1))$
using P **by** *blast*
have 2: $length(inouts_v\ na) = m$
using P **by** *blast*
from 1 2 **have** 3: (*LOR* ($inouts_v\ x$) = *True*)
using P $s1$
by (*metis LOr.elims(3) length-0-conv neq0-conv nth-Cons-0 zero-neq-one*)
show $\exists inouts_v\ inouts_v'$.
 $\forall x. length(inouts_v'\ x) = Suc\ 0 \wedge length(inouts_v\ x) = m \wedge f\text{-LopOR}\ inouts_v\ x = inouts_v'\ x$
apply (*rule-tac* $x = inouts_v$ **in** *exI*)

```

apply (simp add: f-blocks)
apply (rule-tac x =  $\lambda na. [1]$  in exI)
using P 3
by (metis (full-types) LOr.simps(2) hd-Cons-tl length-0-conv length-Cons nth-Cons-0 s1)
next
show  $\forall x na. \text{length}(x na) = m \longrightarrow \text{length}(f\text{-LopOR } x na) = \text{Suc } 0$ 
by (simp add: f-blocks)
qed

```

B.5.11.3 NAND lemma $LNand [1,1] = \text{False}$

by auto

lemma $LNand [1,1,0] = \text{True}$

by auto

lemma $SimBlock\text{-}LopNAND$ [simblock-healthy]:

assumes $s1: m > 0$

shows $SimBlock\ m\ 1\ (LopNAND\ m)$

apply (simp add: f-sim-blocks)

apply (rule $SimBlock\text{-}FBlock$)

proof –

obtain $inouts_v :: nat \Rightarrow \text{real list}$

where $P: \forall na. \text{length}(inouts_v na) = m \wedge (\forall x < m. ((inouts_v na)!x = 0))$

using list-len-avail' **by** fastforce

have 1: $(\forall x < m. ((inouts_v na)!x = 0))$

using P **by** blast

have 2: $\text{length}(inouts_v na) = m$

using P **by** blast

from 1 2 **have** 3: $(LNand (inouts_v x) = \text{True})$

using P s1

by (metis $LNand.elims(3)$ length-0-conv neq0-conv nth-Cons-0)

show $\exists inouts_v inouts_v'$.

$\forall x. \text{length}(inouts_v' x) = \text{Suc } 0 \wedge \text{length}(inouts_v x) = m \wedge f\text{-LopNAND } inouts_v\ x = inouts_v' x$

apply (rule-tac $x = inouts_v$ in exI)

apply (simp add: f-blocks)

apply (rule-tac $x = \lambda na. [1]$ in exI)

using P 3

by (metis (full-types) $LNand.simps(2)$ hd-Cons-tl length-0-conv length-Cons nth-Cons-0 s1)

next

show $\forall x na. \text{length}(x na) = m \longrightarrow \text{length}(f\text{-LopNAND } x na) = \text{Suc } 0$

by (simp add: f-blocks)

qed

B.5.11.4 NOR lemma $LNor [1,0] = \text{False}$

by auto

lemma $LNor [0,0,0] = \text{True}$

by auto

B.5.11.5 XOR lemma $LXor [1,0] 0 = \text{True}$

by auto

lemma $LXor [1,0,1] 0 = \text{False}$

by auto

B.5.11.6 NXOR lemma $LNxor [1,0] 0 = False$
by *auto*

lemma $LNxor [1,0,1] 0 = True$
by *auto*

B.5.11.7 NOT lemma $SimBlock-LopNOT [simblock-healthy]$:
shows $SimBlock 1 1 (LopNOT)$
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*rule-tac x = $\lambda na. [1]$ in exI*)
apply (*simp add: f-LopNOT-def*)
by (*simp add: f-blocks*)

B.5.12 Relational Operator

B.5.12.1 Equal == lemma $SimBlock-RopEQ [simblock-healthy]$:
shows $SimBlock 2 1 (RopEQ)$
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [1]$ in exI*)
apply (*simp add: f-RopEQ-def*)
by (*simp add: f-blocks*)

B.5.12.2 Notequal = lemma $SimBlock-RopNEQ [simblock-healthy]$:
shows $SimBlock 2 1 (RopNEQ)$
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-RopNEQ-def*)
by (*simp add: f-blocks*)

B.5.12.3 Less Than < lemma $SimBlock-RopLT [simblock-healthy]$:
shows $SimBlock 2 1 (RopLT)$
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-RopLT-def*)
by (*simp add: f-blocks*)

B.5.12.4 Less Than or Equal to <= lemma $SimBlock-RopLE [simblock-healthy]$:
shows $SimBlock 2 1 (RopLE)$
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [1]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

B.5.12.5 Greater Than > lemma $SimBlock-RopGT [simblock-healthy]$:
shows $SimBlock 2 1 (RopGT)$

apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

B.5.12.6 Greater Than or Equal to \geq lemma *SimBlock-RopGE* [*simblock-healthy*]:

shows *SimBlock 2 1 (RopGE)*
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,0]$ in exI*)
apply (*rule-tac x = $\lambda na. [1]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

B.5.13 Switch

lemma *SimBlock-Switch1* [*simblock-healthy*]:

shows *SimBlock 3 1 (Switch1 th)*
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,th,1]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

lemma *SimBlock-Switch2* [*simblock-healthy*]:

shows *SimBlock 3 1 (Switch2 th)*
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,th+1,1]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

lemma *SimBlock-Switch3* [*simblock-healthy*]:

shows *SimBlock 3 1 (Switch3)*
apply (*simp add: f-sim-blocks*)
apply (*rule SimBlock-FBlock*)
apply (*rule-tac x = $\lambda na. [0,1,1]$ in exI*)
apply (*rule-tac x = $\lambda na. [0]$ in exI*)
apply (*simp add: f-blocks*)
by (*simp add: f-blocks*)

B.5.14 Merge

B.5.15 Subsystem

B.5.16 Enabled Subsystem

B.5.17 Triggered Subsystem

B.5.18 Enabled and Triggered Subsystem

B.5.19 Data Type Conversion

lemma *SimBlock-DataTypeConvUint32Zero* [*simblock-healthy*]:
 shows *SimBlock* 1 1 (*DataTypeConvUint32Zero*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*rule-tac x = λna. [3294967295.5] in exI*)
 apply (*rule-tac x = λna. [3294967295] in exI*)
 apply (*simp add: f-blocks RoundZero-def uint32-def*)
 by (*simp add: f-blocks*)

lemma *SimBlock-DataTypeConvInt32Zero* [*simblock-healthy*]:
 shows *SimBlock* 1 1 (*DataTypeConvInt32Zero*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*rule-tac x = λna. [-4.5] in exI*)
 apply (*rule-tac x = λna. [-4] in exI*)
 apply (*simp add: f-blocks RoundZero-def int32-def*)
 by (*simp add: f-blocks*)

B.5.20 Initial Condition (IC)

lemma *SimBlock-IC* [*simblock-healthy*]:
 shows *SimBlock* 1 1 (*IC x0*)
 apply (*simp add: f-sim-blocks*)
 apply (*rule SimBlock-FBlock*)
 apply (*rule-tac x = λna. [x0] in exI*)
 apply (*rule-tac x = λna. [x0] in exI*)
 apply (*simp add: f-blocks*)
 by (*simp add: f-blocks*)

B.5.21 Router Block

lemma *assembleOutput-len*:
 $\forall x na. \text{length}(\text{assembleOutput } (x na) \text{ routes}) = \text{length}(\text{routes})$
 apply (*auto*)
 proof (*induction routes*)
 case *Nil*
 then show ?*case*
 by *simp*
 next
 case (*Cons a routes*)
 then show ?*case*
 by (*simp*)
 qed

lemma *SimBlock-Router* [*simblock-healthy*]:
 assumes *s1: length(routes) = m*


```

shows SimBlock m m (Router m routes)
apply (simp add: f-sim-blocks)
apply (rule SimBlock-FBlock)
proof -
  obtain inouts_v::nat  $\Rightarrow$  real list
  where P:  $\forall na. \text{length}(inouts_v na) = m \wedge (\forall x < m. ((inouts_v na)!x = 0))$ 
    using list-len-avail' by fastforce
  have 1:  $(\forall x < m. ((inouts_v na)!x = 0))$ 
    using P by blast
  have 2:  $\text{length}(inouts_v na) = m$ 
    using P by blast
  have 3:  $\forall x. \text{length}(\text{assembleOutput } (inouts_v x) \text{ routes}) = \text{length}(\text{routes})$ 
    by (simp add: assembleOutput-len)
  then have 4:  $\forall x. \text{length}(\text{assembleOutput } (inouts_v x) \text{ routes}) = m$ 
    using s1 by simp
  show  $\exists inouts_v inouts_v'$ .
     $\forall x. \text{length}(inouts_v' x) = m \wedge \text{length}(inouts_v x) = m \wedge f\text{-Router routes } inouts_v x = inouts_v' x$ 
  apply (rule-tac x = inouts_v in exI)
  apply (rule-tac x = f-Rover routes inouts_v in exI)
  apply (simp add: f-blocks)
  using 4 s1
  by (simp add: P)
next
show  $\forall x na. \text{length}(x na) = m \longrightarrow \text{length}(f\text{-Router routes } x na) = m$ 
  apply (simp add: f-blocks)
  using s1 by (simp add: assembleOutput-len)
qed

```

B.6 Frequently Used Composition of Blocks

lemma *UnitDelay-Id-parallel-comp*:

```

(UnitDelay 0 ||B Id) = (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ))
proof -
  have f1: (UnitDelay 0 ||B Id) = (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
    ( $\lambda x n. (((f\text{-UnitDelay } 0) \circ (\lambda xx nn. \text{take } 1 (xx nn))) x n)$ 
      • ( $(f\text{-Id} \circ (\lambda xx nn. \text{drop } 1 (xx nn))) x n$ ))
    using SimBlock-UnitDelay SimBlock-Id apply (simp add: FBlock-parallel-comp f-sim-blocks)
    by (simp add: numeral-2-eq-2)
  then have f1-0: ... = (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
    ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ))
  proof -
    have  $\forall (f::nat \Rightarrow \text{real list}) (n::nat).$ 
      ( $(\lambda x n. (((f\text{-UnitDelay } 0) \circ (\lambda xx nn. \text{take } 1 (xx nn))) x n)$ 
        • ( $(f\text{-Id} \circ (\lambda xx nn. \text{drop } 1 (xx nn))) x n$ )) f n =
        ( $(\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ) f n))
      using f-Id-def f-UnitDelay-def apply (simp)
      by (metis drop-0 drop-Suc list.sel(1) take-Nil take-Suc)
    then show ?thesis
      by auto
  qed
  then show ?thesis
    by (simp add: f1 f1-0)
qed

```

end

C Post Landing Finalize

This is a case study of a subsystem named post landing finalize that is used in aircraft cabin pressure control application. It is from [Honeywell](#) through [D-risQ](#). This case is published in [28] and the diagram of this subsystem is shown in Figure 2 of the paper.

theory *post-landing-finalize-1*

imports

simu-contract-real

simu-contract-real-laws

begin

recall-syntax

sledgehammer-params[

timeout = 200,

verbose = false,

strict = true

]

C.1 Subsystem: *variableTimer*

This subsystem has a rate parameter which is equal to 10.

abbreviation *Rate* $\equiv 10$

This subsystem is composed of two small parts: *variableTimer1* and *variableTimer2*.

abbreviation *variableTimer1* \equiv

$(((((Min2 \;; \; UnitDelay \; 0) \parallel_B (Const \; 1)) \;; \; Sum2) \parallel_B Id \parallel_B (Const \; 0)) \;; \; (Switch1 \; 0.5) \;; \; Split2$

variableTimer1 is simplified by *variableTimer1-simp* to a simple design.

lemma *variableTimer1-simp*:

$variableTimer1 = (FBlock \; (\lambda x \; n. \; True) \; (3) \; 2 \; (\lambda x \; n. \; [if \; (x \; n)!2 \geq 0.5$
 $\quad then \; ((if \; n = 0 \; then \; 0 \; else \; (min \; (hd(x \; (n-1))) \; (hd(tl(x \; (n-1))))) + 1) \; else \; 0,$
 $\quad if \; (x \; n)!2 \geq 0.5$
 $\quad then \; ((if \; n = 0 \; then \; 0 \; else \; (min \; (hd(x \; (n-1))) \; (hd(tl(x \; (n-1))))) + 1) \; else \; 0])))$

proof –

have *f1*: $(Min2 \;; \; UnitDelay \; 0) = (FBlock \; (\lambda x \; n. \; True) \; (2) \; (1) \; ((f-UnitDelay \; 0) \circ f-Min2))$
using *SimBlock-Min2 SimBlock-UnitDelay* **apply** (*simp add: FBlock-parallel-comp f-sim-blocks*)
by (*simp add: FBlock-seq-comp*)

then have *f1-0*: $\dots = (FBlock \; (\lambda x \; n. \; True) \; (2) \; (1) \; (\lambda x \; n. \; [if \; n = 0 \; then \; 0 \; else \; (min \; (hd(x \; (n-1))) \; (hd(tl(x \; (n-1)))))])])$

proof –

have $FBlock \; (\lambda f \; n. \; True) \; 2 \; 1 \; (f-UnitDelay \; 0 \circ f-Min2) = FBlock \; (\lambda f \; n. \; True) \; 2 \; 1$
 $(\lambda f \; n. \; [if \; n = 0 \; then \; 0 \; else \; min \; (hd \; (f \; (n - 1))) \; (hd \; (tl \; (f \; (n - 1))))]) \vee$
 $(\forall f \; n. \; (f-UnitDelay \; 0 \circ f-Min2) \; f \; n = [if \; n = 0 \; then \; 0 \; else$
 $\quad min \; (hd \; (f \; (n - 1))) \; (hd \; (tl \; (f \; (n - 1))))])$

by (*simp add: f-Min2-def f-UnitDelay-def*)

then show *?thesis*

by *meson*

qed

have *simblock-f1*: $SimBlock \; 2 \; 1 \; (FBlock \; (\lambda x \; n. \; True) \; (2) \; (1) \; (\lambda x \; n. \; [if \; n = 0 \; then \; 0 \; else \; (min \; (hd(x \; (n-1))) \; (hd(tl(x \; (n-1)))))])])$

by (*metis (no-types, lifting) Min2-def SimBlock-Min2 SimBlock-FBlock-seq-comp*)

SimBlock-UnitDelay UnitDelay-def f1 f1-0)

have 1: $((\lambda x n. [if\ n = 0\ then\ 0\ else\ (min\ (hd\ (x\ (n-1)))\ (hd\ (tl\ (x\ (n-1))))])) \circ (\lambda xx\ nn. take\ 2\ (xx\ nn)))$
 $= (\lambda x n. [if\ n = 0\ then\ 0\ else\ (min\ (hd\ (x\ (n-1)))\ (hd\ (tl\ (x\ (n-1))))]))$

proof –

have $\forall x n. (((\lambda x n. [if\ n = 0\ then\ 0\ else\ (min\ (hd\ (x\ (n-1)))\ (hd\ (tl\ (x\ (n-1))))])) \circ (\lambda xx\ nn. take\ 2\ (xx\ nn))) x n$
 $= (\lambda x n. [if\ n = 0\ then\ 0\ else\ (min\ (hd\ (x\ (n-1)))\ (hd\ (tl\ (x\ (n-1))))])) x n$

apply (rule allI)+

proof –

fix $x :: 'c \Rightarrow 'd\ list$ **and** $n :: 'c$

have f1: $\forall ds. ds = [] \vee (hd\ ds :: 'd) = ds!(0::nat)$
using *hd-conv-nth* **by** *blast*

have f2: $\neg x\ (n - 1) = [] \longrightarrow \neg take\ 2\ (x\ (n - 1)) = []$
by *simp*

have f3: $take\ (Suc\ 0)\ (tl\ (x\ (n - 1))) = tl\ (take\ (Suc\ (Suc\ 0))\ (x\ (n - 1)))$
by (*simp add: tl-take*)

have f4: $take\ 2\ (x\ (n - 1)) = take\ (Suc\ (Suc\ 0))\ (x\ (n - 1))$
using *numeral-2-eq-2* **by** *presburger*

have f5: $hd\ (tl\ (x\ (n - 1))) = tl\ (x\ (n - 1))!(0::nat) \wedge$
 $hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = tl\ (take\ 2\ (x\ (n - 1)))!(0::nat) \wedge$
 $\neg x\ (n - 1) = [] \longrightarrow min\ (hd\ (take\ 2\ (x\ (n - 1))))$
 $(hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))$

using f3 f2 f1 **by** (*metis One-nat-def less-numeral-extra(1) nth-take numeral-2-eq-2 pos2*)

have f6: $\neg tl\ (take\ 2\ (x\ (n - 1))) = [] \longrightarrow \neg Suc\ 0 = 0 \wedge \neg tl\ (x\ (n - 1)) = []$
using f4 f3 **by** *fastforce*

have f7: $\neg Suc\ 0 = 0$
by *blast*

{ assume $\neg ((\lambda f c. [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))])) \circ (\lambda f c. take\ 2\ (f\ c))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))]$
{ assume $\neg (if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1)))) = min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1))))))$
moreover
{ assume $\neg min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))$
moreover
{ assume $\neg hd\ (take\ 2\ (x\ (n - 1))) = hd\ (x\ (n - 1))$
{ assume $\neg x\ (n - 1) = []$
moreover
{ assume $tl\ (x\ (n - 1)) = [] \wedge hd\ (x\ (n - 1)) = x\ (n - 1)!(0::nat) \wedge hd\ (take\ 2\ (x\ (n - 1))) = take\ 2\ (x\ (n - 1))!(0::nat)$
moreover
{ assume $(tl\ (x\ (n - 1)) = [] \wedge hd\ (x\ (n - 1)) = x\ (n - 1)!(0::nat) \wedge hd\ (take\ 2\ (x\ (n - 1))) = take\ 2\ (x\ (n - 1))!(0::nat) \wedge \neg ((\lambda f c. [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))])) \circ (\lambda f c. take\ 2\ (f\ c))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))]$
moreover
{ assume $(tl\ (x\ (n - 1)) = [] \wedge hd\ (x\ (n - 1)) = x\ (n - 1)!(0::nat) \wedge hd\ (take\ 2\ (x\ (n - 1))) = take\ 2\ (x\ (n - 1))!(0::nat) \wedge \neg (if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1)))) = min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1))))))$
then have $tl\ (take\ 2\ (x\ (n - 1))) = [] \longrightarrow n = 0$
by (*metis (no-types) nth-take pos2*) }
ultimately have $((\lambda f c. [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))])) \circ (\lambda f c. take\ 2\ (f\ c))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \wedge tl$

$(take\ 2\ (x\ (n - 1))) = [] \longrightarrow n = 0$
by fastforce }
ultimately have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \wedge tl\ (take\ 2\ (x\ (n - 1))) = [] \longrightarrow ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))] \vee n = 0$
by blast }
moreover
{ assume $\neg\ tl\ (x\ (n - 1)) = []$
then have $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = []$
using f7 f4 f3 by (metis (no-types) take-eq-Nil) }
ultimately have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \wedge tl\ (take\ 2\ (x\ (n - 1))) = [] \longrightarrow ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))] \vee n = 0$
using f2 f1 by blast }
then have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \wedge tl\ (take\ 2\ (x\ (n - 1))) = [] \longrightarrow ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))] \vee n = 0$
by fastforce }
moreover
{ assume $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = []$
moreover
{ assume $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = [] \wedge \neg\ ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))]$
moreover
{ assume $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = [] \wedge \neg\ (if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1)))) = min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1))))))$
moreover
{ assume $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = [] \wedge \neg\ min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))$
then have $\neg\ tl\ (take\ 2\ (x\ (n - 1))) = [] \wedge \neg\ x\ (n - 1) = []$
by (metis take-eq-Nil)
moreover
{ assume $(hd\ (tl\ (x\ (n - 1)))) = tl\ (x\ (n - 1))!(0::nat) \wedge hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = tl\ (take\ 2\ (x\ (n - 1))!(0::nat) \wedge \neg\ x\ (n - 1) = []) \wedge \neg\ ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))]$
then have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \longrightarrow (hd\ (tl\ (x\ (n - 1)))) = tl\ (x\ (n - 1))!(0::nat) \wedge hd\ (tl\ (take\ 2\ (x\ (n - 1)))) = tl\ (take\ 2\ (x\ (n - 1))!(0::nat) \wedge \neg\ x\ (n - 1) = []) \wedge \neg\ (if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1)))) = min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1))))))$
by fastforce
then have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \longrightarrow n = 0$
using f5 by (metis (no-types)) }
ultimately have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \longrightarrow ((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1))))] \vee n = 0$
using f6 f1 by blast }
ultimately have $((\lambda f\ c.\ [if\ c = 0\ then\ 0\ else\ min\ (hd\ (f\ (c - 1)))\ (hd\ (tl\ (f\ (c - 1))))]) \circ (\lambda f\ c.\ take\ 2\ (f\ c)))\ x\ n = [min\ (hd\ (take\ 2\ (x\ (n - 1))))\ (hd\ (tl\ (take\ 2\ (x\ (n - 1)))))] \longrightarrow n = 0$

$1)))) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))] \longrightarrow$
 $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n =$
 $[\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))] \vee n = 0$
by fastforce }
ultimately have $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))] \longrightarrow$
 $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n =$
 $[\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))] \vee n = 0$
by force }
ultimately have $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))] \longrightarrow ((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))] \vee n = 0$
by blast }
ultimately have $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))] \longrightarrow ((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))] \vee n = 0$
using f3 numeral-2-eq-2 by force }
ultimately have $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))] \longrightarrow ((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))] \vee n = 0$
by presburger }
moreover
{ assume $\neg ((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))]$
then have $\neg [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))]$
 $= [\text{min } (\text{hd } (\text{take } 2 (x (n - 1)))) (\text{hd } (\text{tl } (\text{take } 2 (x (n - 1)))))]$
by simp
then have $n = 0$
by presburger }
ultimately have $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))]$
by fastforce }
then show $((\lambda f c. [\text{if } c = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (f (c - 1))) (\text{hd } (\text{tl } (f (c - 1))))]) \circ (\lambda f c. \text{take } 2 (f c)) x n = [\text{if } n = 0 \text{ then } 0 \text{ else } \text{min } (\text{hd } (x (n - 1))) (\text{hd } (\text{tl } (x (n - 1))))]$
by blast
qed
then show ?thesis
by blast
qed
have $f2: ((\text{Min2 } ; ; \text{UnitDelay } 0) \parallel_B (\text{Const } 1)) =$
 $(\text{FBlock } (\lambda x n. \text{True}) (2) (1))$
 $(\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\text{min } (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1)))))]) \parallel_B (\text{Const } 1)$
using f1 f1-0 by auto
then have $f2-0: \dots = \text{FBlock } (\lambda x n. \text{True}) (2) (2)$
 $(\lambda x n. (((\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\text{min } (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1)))))]) \circ$
 $(\lambda x n. \text{take } 2 (x n))) x n)$

- $(((f-\text{Const } 1) \circ (\lambda x n. \text{drop } 2 (x n)))) x n)$

using SimBlock-Const simblock-f1 apply (simp add: FBlock-parallel-comp f-sim-blocks)
by (simp add: numeral-2-eq-2)
then have $f2-1: \dots = \text{FBlock } (\lambda x n. \text{True}) (2) (2)$
 $(\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\text{min } (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))), 1])$
using 1 f-Const-def by (simp add: 1)

```

have simblock-f2: SimBlock 2 2 (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))], 1]$ ))
  by (metis (no-types, lifting) Const-def SimBlock-Const SimBlock-FBlock-parallel-comp
    Suc-1 Suc-eq-plus1 add-2-eq-Suc f2-0 f2-1 numeral-2-eq-2 simblock-f1)

have f3: (((Min2 ;; UnitDelay 0)  $\parallel_B$  (Const 1)) ;; Sum2) =
  (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
    ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))], 1]$ )) ;; Sum2
  using f2 f2-0 f2-1 by auto
have f3-0: ... = (FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  (f-Sum2 o  $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))], 1]$ ))
  using SimBlock-Sum2 simblock-f2 by (simp add: FBlock-seq-comp f-sim-blocks)
have f3-1: ... = (FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))] + 1]$ ))
  proof -
    have  $\forall x n. ((f\text{-Sum2} \circ (\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))], 1])))$ 
      ( $x n$ )
      = ( $(\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))] + 1]) x n$ )
      by (simp add: f-Sum2-def)
    then show ?thesis
      by presburger
  qed
have simblock-f3: SimBlock 2 1 (FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))] + 1]$ ))
  by (metis (no-types, lifting) SimBlock-FBlock-seq-comp SimBlock-Sum2 Sum2-def f3-0 f3-1 simblock-f2)

have f4: (Id  $\parallel_B$  (Const 0)) = (FBlock ( $\lambda x n. \text{True}$ ) (1) (2)
  ( $\lambda x n. (((f\text{-Id} \circ (\lambda x n n. \text{take } 1 (x n))) x n) \bullet (((f\text{-Const } 0) \circ (\lambda x n n. \text{drop } 1 (x n)))) x n))$ )
  using SimBlock-Const SimBlock-Id apply (simp add: FBlock-parallel-comp f-sim-blocks)
  by (simp add: numeral-2-eq-2)
then have f4-0: ... = FBlock ( $\lambda x n. \text{True}$ ) 1 2 ( $\lambda x n. [\text{hd}(x n), 0]$ )
  proof -
    have  $\forall x n. ((\lambda x n. (((f\text{-Id} \circ (\lambda x n n. \text{take } 1 (x n))) x n) \bullet$ 
      ( $((f\text{-Const } 0) \circ (\lambda x n n. \text{drop } 1 (x n))) x n$ )) x n)
      = ( $(\lambda x n. [\text{hd}(x n), 0]) x n$ )
      by (smt append.left-neutral append-Cons append-take-drop-id comp-apply f-Const-def
        f-Id-def hd-append2 take-eq-Nil zero-neq-one)
    then show ?thesis
      by presburger
  qed
have simblock-f4: SimBlock (Suc 0) 2 (FBlock ( $\lambda x n. \text{True}$ ) (Suc 0) 2 ( $\lambda x n. [\text{hd}(x n), 0]$ ))
  using SimBlock-Const SimBlock-Id SimBlock-FBlock-seq-comp
  by (metis (no-types, lifting) Const-def Id-def One-nat-def SimBlock-FBlock-parallel-comp
    Suc-eq-plus1-left f4 f4-0 nat-1-add-1)

have f5: ((((Min2 ;; UnitDelay 0)  $\parallel_B$  (Const 1)) ;; Sum2)  $\parallel_B$  Id) =
  (FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
    ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))] + 1]$ )  $\parallel_B$  Id
  using f3 f3-0 f3-1 by auto
then have f5-0: ... =
  (FBlock ( $\lambda x n. \text{True}$ ) (3) (2)
    ( $\lambda x n. (((\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } (\min (\text{hd}(x (n-1))) (\text{hd}(\text{tl}(x (n-1))))] + 1])$ 
      o  $(\lambda x n n. \text{take } 2 (x n))) x n$ )
      • ( $(f\text{-Id} \circ (\lambda x n n. \text{drop } 2 (x n))) x n$ ))
  using simblock-f3 SimBlock-Id apply (simp add: FBlock-parallel-comp f-sim-blocks)

```

by (*simp add: numeral-2-eq-2*)
then have *f5-1*: ... =
 (*FBlock* ($\lambda x n. \text{True}$) (*3*) (*2*))
 ($\lambda x n. [(if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1)))))) + 1, (x\ n)!2]$)
proof –
have *11*: $\forall inouts_v\ x. (min\ (hd\ (take\ 2\ (inouts_v\ (x - Suc\ 0))))\ (hd\ (tl\ (take\ 2\ (inouts_v\ (x - Suc\ 0)))))) + 1$
 = $min\ (hd\ (inouts_v\ (x - Suc\ 0)))\ (hd\ (tl\ (inouts_v\ (x - Suc\ 0)))) + 1$
by (*smt Suc-1 append-take-drop-id diff-Suc-1 hd-append2 take-eq-Nil tl-take zero-neq-one zero-not-eq-two*)
have *12*: $\forall inouts_v\ x. (length(inouts_v\ x) = 3 \longrightarrow$
 (*f-Id* ($\lambda nn. drop\ 2\ (inouts_v\ nn)$) *x*) = [*inouts_v* *x*!(*2*)])
by (*simp add: f-Id-def hd-drop-conv-nth*)
have *2*: $\forall inouts_v\ x. (length(inouts_v\ x) = 3 \longrightarrow$
 (((*min* (*hd* (*take* 2 (*inouts_v* (*x - Suc* 0)))) (*hd* (*tl* (*take* 2 (*inouts_v* (*x - Suc* 0)))))) + 1) #
f-Id ($\lambda nn. drop\ 2\ (inouts_v\ nn)$) *x*)
 = [*min* (*hd* (*inouts_v* (*x - Suc* 0))) (*hd* (*tl* (*inouts_v* (*x - Suc* 0)))) + 1, *inouts_v* *x*!(*2*)])
using *11 12* **by** *blast*
show ?*thesis*
apply (*simp add: FBlock-def*)
apply (*rel-auto*)
apply (*metis (no-types, lifting) One-nat-def Suc-1 f-Id-def hd-drop-conv-nth lessI numeral-3-eq-3*)
using *11 12 2*
apply *metis*
apply (*simp add: 12*)
apply (*simp add: 11 12*)
by (*simp add: f-Id-def*)
qed
have *simblock-f5*: *SimBlock* 3 2 (*FBlock* ($\lambda x n. \text{True}$) (*3*) (*2*))
 ($\lambda x n. [(if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1)))))) + 1, (x\ n)!2]$)
by (*smt Id-def SimBlock-Id SimBlock-FBlock-parallel-comp add commute f5-0 f5-1 one-add-one one-plus-numeral semiring-norm(3) simblock-f3*)

have *f6*: (((*Min2* ;; *UnitDelay* 0) $\|_B$ (*Const* 1)) ;; *Sum2*) $\|_B$ *Id* $\|_B$ (*Const* 0))
 = (*FBlock* ($\lambda x n. \text{True}$) (*3*) (*2*))
 ($\lambda x n. [(if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1)))))) + 1, (x\ n)!2]$)
 $\|_B$ (*Const* 0)
using *f5 f5-0 f5-1* **by** *auto*
then have *f6-0*: ... = (*FBlock* ($\lambda x n. \text{True}$) (*3*) (*3*))
 ($\lambda x n. (((\lambda x n. [(if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1)))))) + 1, (x\ n)!2]$)
 $\circ (\lambda xx\ nn. take\ 3\ (xx\ nn)))\ x\ n$)
 • (((*f-Const* 0) $\circ (\lambda xx\ nn. drop\ 3\ (xx\ nn)))\ x\ n))$)
using *simblock-f5 SimBlock-Const* **by** (*simp add: FBlock-parallel-comp f-sim-blocks*)
then have *f6-1*: ... = (*FBlock* ($\lambda x n. \text{True}$) (*3*) (*3*))
 ($\lambda x n. [(if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1)))))) + 1, (x\ n)!2, 0]$)
proof –
have *11*: $\forall inouts_v\ x. ((f-Const\ 0\ (\lambda nn. drop\ 3\ (inouts_v\ nn))\ x)) = [0]$
by (*simp add: f-Const-def*)
have *12*: $\forall inouts_v\ x. length(inouts_v\ x) = 3 \longrightarrow (take\ 3\ (inouts_v\ x)) = inouts_v\ x$
by *simp*

show *FBlock* ($\lambda x n. \text{True}$) 3 3
 ($\lambda x n. ((\lambda x n. [(if\ n = 0\ then\ 0\ else\ min\ (hd\ (x\ (n - 1)))\ (hd\ (tl\ (x\ (n - 1)))) + 1, x$
n!(*2*)] \circ
 ($\lambda xx\ nn. take\ 3\ (xx\ nn)))\ x\ n \bullet (f-Const\ 0 \circ (\lambda xx\ nn. drop\ 3\ (xx\ nn)))\ x\ n$))

$= \text{FBlock } (\lambda x n. \text{True}) \ 3 \ 3 \ (\lambda x n. [(if \ n = 0 \ \text{then} \ 0 \ \text{else} \ \min \ (\text{hd} \ (x \ (n - 1))) \ (\text{hd} \ (\text{tl} \ (x \ (n - 1))))]) + 1, \ x \ n!(2), \ 0]$
apply (*simp add: FBlock-def*)
apply (*rel-auto*)
apply (*simp add: f-Const-def*)
proof –
fix ok_v **and** $inouts_v::nat \Rightarrow \text{real list}$ **and** ok_v' **and** $inouts_v'::nat \Rightarrow \text{real list}$ **and** $x::nat$
assume $a1: \forall x. (x = 0 \longrightarrow$
 $\text{length}(inouts_v \ 0) = 3 \wedge$
 $\text{length}(inouts_v' \ 0) = 3 \wedge 1 \# inouts_v \ 0!(2) \# f\text{-Const } 0 \ (\lambda nn. \text{drop } 3 \ (inouts_v \ nn)) \ 0 =$
 $inouts_v' \ 0) \wedge$
 $(0 < x \longrightarrow$
 $\text{length}(inouts_v \ x) = 3 \wedge$
 $\text{length}(inouts_v' \ x) = 3 \wedge$
 $(\min \ (\text{hd} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))) \ (\text{hd} \ (\text{tl} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))))) + 1) \#$
 $inouts_v \ x!(2) \# f\text{-Const } 0 \ (\lambda nn. \text{drop } 3 \ (inouts_v \ nn)) \ x =$
 $inouts_v' \ x)$
assume $a2: 0 < x$
from $a1$ **have** $1: \forall x. \text{length}(inouts_v \ x) = 3$
using *gr0I* **by** *blast*
from $a2 \ a1$ **have** $2:$
 $(\min \ (\text{hd} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))) \ (\text{hd} \ (\text{tl} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))))) + 1) \#$
 $inouts_v \ x!(2) \# f\text{-Const } 0 \ (\lambda nn. \text{drop } 3 \ (inouts_v \ nn)) \ x = inouts_v' \ x$
by *blast*
from $a2 \ 1$ **have** $3: \text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)) = inouts_v \ (x - \text{Suc } 0)$
by *simp*
show $[\min \ (\text{hd} \ (inouts_v \ (x - \text{Suc } 0))) \ (\text{hd} \ (\text{tl} \ (inouts_v \ (x - \text{Suc } 0)))) + 1, inouts_v \ x!(2),$
 $0] = inouts_v' \ x$
by (*metis 1 11 2 order-refl take-all*)
next
fix ok_v **and** $inouts_v::nat \Rightarrow \text{real list}$ **and** ok_v' **and** $inouts_v'::nat \Rightarrow \text{real list}$
assume $a1: \forall x. (x = 0 \longrightarrow \text{length}(inouts_v \ 0) = 3 \wedge \text{length}(inouts_v' \ 0) = 3 \wedge [1, inouts_v$
 $0!(2), 0] = inouts_v' \ 0) \wedge$
 $(0 < x \longrightarrow$
 $\text{length}(inouts_v \ x) = 3 \wedge$
 $\text{length}(inouts_v' \ x) = 3 \wedge$
 $[\min \ (\text{hd} \ (inouts_v \ (x - \text{Suc } 0))) \ (\text{hd} \ (\text{tl} \ (inouts_v \ (x - \text{Suc } 0)))) + 1, inouts_v \ x!(2), 0] =$
 $inouts_v' \ x)$
show $1 \# inouts_v \ 0!(2) \# f\text{-Const } 0 \ (\lambda nn. \text{drop } 3 \ (inouts_v \ nn)) \ 0 = inouts_v' \ 0$
by (*simp add: 11 a1*)
next
fix ok_v **and** $inouts_v::nat \Rightarrow \text{real list}$ **and** ok_v' **and** $inouts_v'::nat \Rightarrow \text{real list}$ **and** $x::nat$
assume $a1: \forall x. (x = 0 \longrightarrow \text{length}(inouts_v \ 0) = 3 \wedge \text{length}(inouts_v' \ 0) = 3 \wedge [1, inouts_v$
 $0!(2), 0] = inouts_v' \ 0) \wedge$
 $(0 < x \longrightarrow$
 $\text{length}(inouts_v \ x) = 3 \wedge$
 $\text{length}(inouts_v' \ x) = 3 \wedge$
 $[\min \ (\text{hd} \ (inouts_v \ (x - \text{Suc } 0))) \ (\text{hd} \ (\text{tl} \ (inouts_v \ (x - \text{Suc } 0)))) + 1, inouts_v \ x!(2), 0] =$
 $inouts_v' \ x)$
assume $a2: x > 0$
from $a1$ **have** $1: \forall x. \text{length}(inouts_v \ x) = 3$
using *gr0I* **by** *blast*
from $a2 \ 1$ **have** $3: \text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)) = inouts_v \ (x - \text{Suc } 0)$
by *simp*
show $(\min \ (\text{hd} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))) \ (\text{hd} \ (\text{tl} \ (\text{take } 3 \ (inouts_v \ (x - \text{Suc } 0)))))) +$

1) #

```

    inoutsv x!(2) # f-Const 0 (λnn. drop 3 (inoutsv nn)) x =
    inoutsv' x
  by (simp add: 11 3 a1 a2)
next
  fix okv and inoutsv::nat⇒real list and okv' and inoutsv'::nat⇒real list
  and x::nat⇒real list and xa::nat
  show length(f-Const 0 (λnn. drop 3 (x nn)) xa) = Suc 0
  by (simp add: f-Const-def)
qed
qed
have simblock-f6: SimBlock 3 3 (FBlock (λx n. True) (3) (3)
  (λx n. [(if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1, (x n)!2, 0]))
  using Const-def simblock-f5 SimBlock-FBlock-parallel-comp
  by (metis (no-types, lifting) One-nat-def SimBlock-Const Suc3-eq-add-3 add commute
    add-2-eq-Suc' f6-0 f6-1 numeral-3-eq-3)

have f7: (((Min2 ;; UnitDelay 0) ||B (Const 1)) ;; Sum2) ||B Id ||B (Const 0) ;; (Switch1 0.5)
  = (FBlock (λx n. True) (3) (3) (λx n. [(if n = 0 then 0 else
    (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1, (x n)!2, 0])) ;; (Switch1 0.5)
  using f6 f6-0 f6-1 by auto
have f7-0: ... = (FBlock (λx n. True) (3) 1 ((f-Switch1 0.5) o (λx n. [(if n = 0 then 0 else
    (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1, (x n)!2, 0]))))
  using simblock-f6 SimBlock-Switch1 by (simp add: FBlock-seq-comp Switch1-def)
have f7-1: ... = FBlock (λx n. True) (3) 1
  (λx n. [if (x n)!2 ≥ 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1)
    else 0])
  proof -
  have 1: ∀ x n. (((f-Switch1 0.5) o (λx n. [(if n = 0 then 0 else
    (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1, (x n)!2, 0])) x n
  =
  (λx n. [if (x n)!2 ≥ 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1)
    else 0]) x n)
  apply (auto)
  by (simp add: f-Switch1-def)+
  then show ?thesis
  by presburger
qed
have simblock-f7: SimBlock 3 1 (FBlock (λx n. True) (3) 1
  (λx n. [if (x n)!2 ≥ 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0]))
  using simblock-f6 SimBlock-Switch1 SimBlock-FBlock-seq-comp f7 f7-0 f7-1
  by (metis (no-types, lifting) Switch1-def)

have f8: (((Min2 ;; UnitDelay 0) ||B (Const 1)) ;; Sum2) ||B Id ||B (Const 0) ;;
  (Switch1 0.5) ;; Split2 =
  ((FBlock (λx n. True) (3) 1 (λx n. [if (x n)!2 ≥ 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0])) ;; Split2)
  by (metis RA1 f7 f7-0 f7-1)
have f8-0: ... = (FBlock (λx n. True) (3) 2 (f-Split2 o (λx n. [if (x n)!2 ≥ 0.5
  then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0]))))
  using simblock-f7 SimBlock-Split2
  by (simp add: FBlock-seq-comp Split2-def)

```

have *f8-1*: ... = (FBlock (λx n. True) (3) 2 (λx n. [if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0]))
proof –
have *11*: ∀ x n. ((f-Split2 o (λx n. [if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0])) x n)
= (λx n. [if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0]) x n
apply (auto)
by (simp add: f-Split2-def)+
show ?thesis
using *11* **by** presburger
qed
have *simblock-f8*: SimBlock 3 2 (FBlock (λx n. True) (3) 2 (λx n. [if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
if (x n)!2 ≥ 0.5
then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0]))
using *simblock-f7 f8 f8-0 f8-1 SimBlock-Split2*
by (metis (no-types, lifting) SimBlock-FBlock-seq-comp Split2-def)
show ?thesis
using *f8 f8-0 f8-1* **by** auto
qed

abbreviation *variableTimer2* ≡
((Const 0) ||_B Id) ;; Max2 ;; (Gain Rate) ;; RoundCeil ;; DataTypeConvInt32Zero ;; Split2

variableTimer2 is also simplified by *variableTimer2-simp*.

lemma *variableTimer2-simp*:

variableTimer2 = (FBlock (λx n. True) (Suc 0) (2)
(λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))),
real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))]))

proof –

have *f1*: ((Const 0) ||_B Id) = (FBlock (λx n. True) (1) (2)
(λx n. (((f-Const 0) o (λxx nn. take 0 (xx nn))) x n) • ((f-Id o (λxx nn. drop 0 (xx nn)))) x n))
using *SimBlock-Const SimBlock-Id* **apply** (simp add: FBlock-parallel-comp f-sim-blocks)
by (simp add: numeral-2-eq-2)
then have *f1-0*: ... = FBlock (λx n. True) (Suc 0) 2 (λx n. [0, hd(x n)])
by (simp add: f-blocks)
have *simblock-f1*: SimBlock (Suc 0) 2 (FBlock (λx n. True) (Suc 0) 2 (λx n. [0, hd(x n)]))
using *SimBlock-Const SimBlock-Id SimBlock-FBlock-seq-comp*
by (metis (no-types, lifting) *f1 f1-0 Const-def Id-def SimBlock-FBlock-parallel-comp Suc-eq-plus1 nat-1-add-1*)

have *f2*: ((Const 0) ||_B Id) ;; Max2 = FBlock (λx n. True) (Suc 0) 2 (λx n. [0, hd(x n)]) ;;
Max2

using *f1-0* **by** (simp add: *f1*)
have *f2-0*: ... = FBlock (λx n. True) (Suc 0) (Suc 0) (f-Max2 o (λx n. [0, hd(x n)]))
using *simblock-f1 SimBlock-Max2* **by** (simp add: FBlock-seq-comp f-sim-blocks)
have *f2-1*: ... = FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [max (hd(x n)) 0])
using *f-Max2-def*

```

    by (metis (mono-tags, lifting) comp-eq-dest-lhs list.sel(1) list.sel(3) max.commute)
  have simblock-f2: SimBlock (Suc 0) (Suc 0) (FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [max
(hd(x n)) 0]))
    using simblock-f1 SimBlock-Max2 SimBlock-FBlock-seq-comp
    by (metis Max2-def One-nat-def f2-0 f2-1)
  have f3: ((Const 0) ||B Id) ;; Max2 ;; (Gain Rate) =
    (FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [max (hd(x n)) 0])) ;; (Gain Rate)
    using f2-1 f2-0 by (simp add: RA1 f2)
  then have f3-0: ... = FBlock (λx n. True) (Suc 0) (Suc 0) ((f-Gain Rate) o (λx n. [max (hd(x n))
0]))
    using SimBlock-Gain simblock-f2 by (simp add: FBlock-seq-comp f-sim-blocks)
  then have f3-1: ... = FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [Rate * (max (hd(x n)) 0)])
  proof -
    have ∀ f n. (f-Gain Rate) o (λf n. [max (hd (f n)) 0])) f n = [Rate * max (hd (f n)) 0]
      by (simp add: f-Gain-def)
    then show ?thesis
      by presburger
  qed
  have simblock-f3: SimBlock (Suc 0) (Suc 0)
    (FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [Rate * (max (hd(x n)) 0)]))
    using simblock-f2 SimBlock-Gain SimBlock-FBlock-seq-comp
    by (metis Gain-def One-nat-def f3-0 f3-1)

  have f4: ((Const 0) ||B Id) ;; Max2 ;; (Gain Rate) ;; RoundCeil =
    (FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [Rate * (max (hd(x n)) 0)])) ;; RoundCeil
    using f3-0 f3-1 by (simp add: RA1 f2 f2-0 f2-1)
  then have f4-0: ... = (FBlock (λx n. True) (Suc 0) (Suc 0) (
    (f-RoundCeil) o (λx n. [Rate * (max (hd(x n)) 0)])))
    using SimBlock-RoundCeil simblock-f3 by (simp add: FBlock-seq-comp RoundCeil-def)
  then have f4-1: ... = (FBlock (λx n. True) (Suc 0) (Suc 0) (
    (λx n. [real-of-int [Rate * (max (hd(x n)) 0)]]))))
  proof -
    have ∀ f n. (f-RoundCeil) o (λf n. [Rate * max (hd (f n)) 0])) f n = [real-of-int [Rate * max (hd
(f n)) 0]]
      by (simp add: f-RoundCeil-def)
    then show ?thesis
      by presburger
  qed
  have simblock-f4: SimBlock (Suc 0) (Suc 0)
    (FBlock (λx n. True) (Suc 0) (Suc 0) ((λx n. [real-of-int [Rate * (max (hd(x n)) 0)]]))))
    using simblock-f3 SimBlock-RoundCeil SimBlock-FBlock-seq-comp
    by (metis One-nat-def RoundCeil-def f4-0 f4-1)

  have f5: ((Const 0) ||B Id) ;; Max2 ;; (Gain Rate) ;; RoundCeil ;; DataTypeConvInt32Zero
    = (FBlock (λx n. True) (Suc 0) (Suc 0) (λx n. [real-of-int [Rate * (max (hd(x n)) 0)]]))
      ;; DataTypeConvInt32Zero
    by (metis RA1 f4 f4-0 f4-1)
  then have f5-0: ... = (FBlock (λx n. True) (Suc 0) (Suc 0)
    (f-DTConvInt32Zero o (λx n. [real-of-int [Rate * (max (hd(x n)) 0)]]))))
    by (metis DataTypeConvInt32Zero-def One-nat-def FBlock-seq-comp
    SimBlock-DataTypeConvInt32Zero simblock-f4)
  then have f5-1: ... = (FBlock (λx n. True) (Suc 0) (Suc 0)
    (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)])))]))
  proof -
    have ∀ f n. (f-DTConvInt32Zero) o (λf n. [real-of-int [(Rate::real) * max (hd (f n)) 0]])) f n

```

```

    = [real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (f n)) 0])))]
  by (simp add: f-DTConvInt32Zero-def)
then show ?thesis
  by presburger
qed
have simblock-f5: SimBlock (Suc 0) (Suc 0) ((FBlock (λx n. True) (Suc 0) (Suc 0)
  (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))))))
  by (metis DataTypeConvInt32Zero-def One-nat-def SimBlock-DataTypeConvInt32Zero
    SimBlock-FBlock-seq-comp f5-0 f5-1 simblock-f4)

have f6: ((Const 0) ||B Id) ;; Max2 ;; (Gain Rate) ;; RoundCeil ;; DataTypeConvInt32Zero ;;
Split2
  = ((FBlock (λx n. True) (Suc 0) (Suc 0)
    (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))))))
    ;; Split2
  by (metis RA1 f5 f5-0 f5-1)
then have f6-0: ... = (FBlock (λx n. True) (Suc 0) (2)
  (f-Split2 o (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))))))
  by (metis Split2-def One-nat-def FBlock-seq-comp
    SimBlock-Split2 simblock-f5)
then have f6-1: ... = (FBlock (λx n. True) (Suc 0) (2)
  (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)])))]))
proof -
  have ∀ f n. [real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (hd (f n)) 0]))),
    real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (f n)) 0])))] =
    (f-Split2 o (λf n. [real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (f n)) 0]))])) f n
  by (simp add: f-Split2-def)
  then show ?thesis
    by presburger
qed
have simblock-f6: SimBlock 1 2 (FBlock (λx n. True) (Suc 0) (2)
  (λx n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)])))]))
  by (metis (no-types, lifting) One-nat-def SimBlock-FBlock-seq-comp SimBlock-Split2
    Split2-def f6-0 f6-1 simblock-f5)
show ?thesis
  by (simp add: f6 f6-0 f6-1)
qed

```

The *variableTimer* subsystem is composed of two parts by means of parallel composition and feedback.

definition *variableTimer* ≡

((*variableTimer1* ||_B *variableTimer2*) *f_D* (0,0)) *f_D* (0,2)) ;; *RopGT*

vT-fd-sol-1 calculates the output from its current and past inputs recursively. It is a solution for the first feedback in *variableTimer*.

fun *vT-fd-sol-1*:: (nat ⇒ real) ⇒ (nat ⇒ real) ⇒ nat ⇒ real **where**

vT-fd-sol-1 *door-open-time* *door-open* 0 =

(if *door-open* 0 ≥ 0.5 then 1.0 else 0) |

vT-fd-sol-1 *door-open-time* *door-open* (Suc n) =

(if *door-open* (Suc n) ≥ 0.5

then ((min (*vT-fd-sol-1* *door-open-time* *door-open* n) (*door-open-time* n)) + 1)

else 0)

vT-fd-sol-1 is proved to be a solution for the first feedback. This lemma will be used later to expand the first feedback.

lemma *vT-fd-sol-1-is-a-solution*:

```

fixes inouts0::nat  $\Rightarrow$  real list and n::nat
assumes a1:  $\forall x. \text{length}(\text{inouts}_0\ x) = 3$ 
shows  $0 < n \longrightarrow (1 \leq \text{inouts}_0\ n!(\text{Suc}\ 0) * 2 \longrightarrow$ 
   $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n =$ 
   $\text{min}\ (vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ (n - \text{Suc}\ 0))$ 
   $(\text{hd}\ (\text{inouts}_0\ (n - \text{Suc}\ 0))) + 1) \wedge$ 
   $(\neg\ 1 \leq \text{inouts}_0\ n!(\text{Suc}\ 0) * 2 \longrightarrow$ 
   $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n = 0)$ 
apply (clarify, rule conjI, clarify)
defer
apply (clarify)
proof -
  assume a1:  $0 < n$ 
  assume a2:  $\neg\ 1 \leq \text{inouts}_0\ n!(\text{Suc}\ 0) * 2$ 
  from a2 have a2':  $\text{inouts}_0\ n!(\text{Suc}\ 0) < 0.5$ 
    by (simp)
  have 1:  $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n$ 
     $= vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ (\text{Suc}\ (n - \text{Suc}\ 0))$ 
    using a1 by simp
  show  $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n = 0$ 
    apply (simp add: 1)
    using a2' by (simp add: a1)
next
  assume a1:  $0 < n$ 
  assume a2:  $1 \leq \text{inouts}_0\ n!(\text{Suc}\ 0) * 2$ 
  from a2 have a2':  $\text{inouts}_0\ n!(\text{Suc}\ 0) \geq 0.5$ 
    by (simp)
  have 1:  $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n$ 
     $= vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ (\text{Suc}\ (n - \text{Suc}\ 0))$ 
    using a1 by simp
  show  $vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ n =$ 
     $\text{min}\ (vT\text{-fd-sol-1}\ (\lambda n1. \text{hd}\ (\text{inouts}_0\ n1))\ (\lambda n1. \text{inouts}_0\ n1!(\text{Suc}\ 0))\ (n - \text{Suc}\ 0))$ 
     $(\text{hd}\ (\text{inouts}_0\ (n - \text{Suc}\ 0))) + 1$ 
    apply (simp add: 1)
    using a2' a1 by simp
qed

```

variableTimer-simp-pat-f gives the function definition of the finally simplified subsystem.

abbreviation *variableTimer-simp-pat-f*

```

 $\equiv (\lambda x\ na. [\text{if}\ (\text{if}\ 1 \leq x\ na!(0) * 2$ 
   $\text{then}\ (\text{if}\ na = 0\ \text{then}\ 0$ 
     $\text{else}\ \text{min}\ (vT\text{-fd-sol-1}$ 
       $(\lambda n1. (\lambda na. \text{real-of-int}$ 
         $(\text{int32}\ (\text{RoundZero}\ (\text{real-of-int}\ [\text{Rate}\ * \text{max}\ (x\ na!(\text{Suc}\ 0))\ 0])))\ n1)$ 
         $(\lambda n1. (x\ n1)!(0))\ (na - 1))$ 
         $((\lambda na. \text{real-of-int}\ (\text{int32}\ (\text{RoundZero}\ (\text{real-of-int}\ [\text{Rate}\ * \text{max}\ (x\ na!(\text{Suc}\ 0))\ 0])))$ 
           $(na - 1))) + 1$ 
       $\text{else}\ 0) > (\text{real-of-int}\ (\text{int32}\ (\text{RoundZero}\ (\text{real-of-int}\ [\text{Rate}\ * \text{max}\ (x\ na!(\text{Suc}\ 0))\ 0])))$ 
         $\text{then}\ 1\ \text{else}\ 0])$ 

```

variableTimer-simp-pat is the simplified block for the subsystem.

abbreviation *variableTimer-simp-pat*

\equiv (*FBlock* ($\lambda x n. \text{True}$) (2) 1 *variableTimer-simp-pat-f*)

variableTimer-simp-pat is also a block.

lemma *SimBlock-variableTimer-simp*:

SimBlock 2 1 *variableTimer-simp-pat*
apply (*rule SimBlock-FBlock*)
apply (*rule-tac* $x = \lambda na. [0, 0]$ **in** exI)
apply (*rule-tac* $x = \lambda na. [0]$ **in** exI)
apply (*simp*)
apply (*simp add: int32-def RoundZero-def*)
by *simp*

variableTimer-simp simplifies the subsystem into a block.

lemma *variableTimer-simp*:

variableTimer = *variableTimer-simp-pat*

proof –

let $?vt-f = (\lambda x na. [if (if 1 \leq x na!(0) * 2$
 $then (if na = 0 then 0$
 $else min (vT-fd-sol-1$
 $(\lambda n1. (\lambda na. real-of-int$
 $(int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1$
 $(\lambda n1. (x n1)!(0)) (na - 1))$
 $((\lambda na. real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))$
 $(na - 1)))] + 1$
 $else 0) > (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))$
 $then 1 else 0])$

have *simblock-variableTimer1*: *SimBlock* 3 2 (*FBlock* ($\lambda x n. \text{True}$) (3) 2 ($\lambda x n. [if (x n)!2 \geq 0.5$
 $then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,$
 $if (x n)!2 \geq 0.5$

$then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0])$

apply (*simp add: SimBlock-def FBlock-def*)

apply (*rel-auto*)

apply (*rule-tac* $x = \lambda na. [2, 1, 0.51]$ **in** exI , *simp*)

apply (*rule-tac* $x = \lambda na. (if na = 0 then [1,1] else [2,2])$ **in** exI)

by (*simp*)

have *simblock-variableTimer2*: *SimBlock* (*Suc* 0) 2 (*FBlock* ($\lambda x n. \text{True}$) (*Suc* 0) (2)
 $(\lambda x n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))],$
 $real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))]))$

apply (*simp add: SimBlock-def FBlock-def*)

apply (*rel-auto*)

apply (*rule-tac* $x = \lambda na. [1]$ **in** exI , *simp*)

apply (*rule-tac* $x = \lambda na. [Rate, Rate]$ **in** exI , *simp*)

by (*simp add: RoundZero-def int32-def*)

have *f1*: (*variableTimer1* \parallel_B *variableTimer2*)

= (*FBlock* ($\lambda x n. \text{True}$) (3) 2 ($\lambda x n. [if (x n)!2 \geq 0.5$

$then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,$

$if (x n)!2 \geq 0.5$

$then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0])$

\parallel_B

(*FBlock* ($\lambda x n. \text{True}$) (*Suc* 0) (2)

$(\lambda x n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))],$

$real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))]))$

using *variableTimer1-simp variableTimer2-simp* **by** *auto*

then have *f1-0*: ... = (*FBlock* ($\lambda x n. \text{True}$) (4) 4

```

( $\lambda x n. (((\lambda x n.
  [if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0])
  \circ (\lambda x n n. take 3 (x n)) x n)
  \bullet (((\lambda x n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))])
  \circ (\lambda x n n. drop 3 (x n)) x n))
using simblock-variableTimer1 simblock-variableTimer2 by (simp add: FBlock-parallel-comp f-sim-blocks)
then have f1-1: ... = (FBlock ( $\lambda x n. True$ ) (4) 4)
( $\lambda x n.
  [if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3 0)]))])])
proof -
have 11:  $\forall x n. ((length(x n) = 4) \longrightarrow ((\lambda x n. (((\lambda x n.
  [if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0])
  \circ (\lambda x n n. take 3 (x n)) x n)
  \bullet (((\lambda x n. [real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max (hd(x n)) 0)]))])
  \circ (\lambda x n n. drop 3 (x n)) x n)) x n)
= ((\lambda x n.
  [if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    if (x n)!2 \geq 0.5
    then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1)))))) + 1) else 0,
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3 0)]))),
    real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3 0)]))]) x n)
  apply (auto)
  apply (simp add: hd-drop-conv-nth)
  apply (smt diff-Suc-1 hd-conv-nth list.sel(2) nth-take numeral-3-eq-3 take-eq-Nil tl-take
    zero-less-Suc zero-neq-numeral)
apply (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
  by (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
show ?thesis
  apply (simp add: FBlock-def)
  apply (rel-simp)
  apply (rule iffI)
  apply (clarify)
  apply (rule conjI)
  apply (clarify)
  apply (rule conjI)
  apply (clarify)
apply (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
apply (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
  apply (clarify)
  apply (rule conjI)
  apply (clarify)$$$ 
```

```

apply (rule conjI)
apply blast
apply (rule conjI)
apply blast
proof -
  fix  $ok_v$  and  $inouts_v::nat \Rightarrow real\ list$  and  $ok_v'$  and  $inouts_v':::nat \Rightarrow real\ list$ 
    and  $x::nat$ 
  assume  $a1: \forall x. (x = 0 \longrightarrow$ 
     $(1 \leq inouts_v\ 0!(2) * 2 \longrightarrow$ 
       $length(inouts_v\ 0) = 4 \wedge$ 
       $length(inouts_v'\ 0) = 4 \wedge$ 
       $[1, 1, real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ 0)))$ 
0]))),
```

$real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ 0)))\ 0]))] =$
 $inouts_v'\ 0) \wedge$
 $(\neg 1 \leq inouts_v\ 0!(2) * 2 \longrightarrow$
 $length(inouts_v\ 0) = 4 \wedge$
 $length(inouts_v'\ 0) = 4 \wedge$
 $[0, 0, real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ 0)))$
0]))),

$real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ 0)))\ 0]))] =$
 $inouts_v'\ 0) \wedge$
 $(0 < x \longrightarrow$
 $(1 \leq inouts_v\ x!(2) * 2 \longrightarrow$
 $length(inouts_v\ x) = 4 \wedge$
 $length(inouts_v'\ x) = 4 \wedge$
 $[min (hd (take\ 3\ (inouts_v\ (x - Suc\ 0)))) (hd (tl (take\ 3\ (inouts_v\ (x - Suc\ 0)))))) + 1,$
 $min (hd (take\ 3\ (inouts_v\ (x - Suc\ 0)))) (hd (tl (take\ 3\ (inouts_v\ (x - Suc\ 0)))))) + 1,$
 $real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ x)))\ 0]))),$
 $real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ x)))\ 0]))] =$
 $inouts_v'\ x) \wedge$
 $(\neg 1 \leq inouts_v\ x!(2) * 2 \longrightarrow$
 $length(inouts_v\ x) = 4 \wedge$
 $length(inouts_v'\ x) = 4 \wedge$
 $[0, 0, real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ x)))$
0]))),

$real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (hd (drop\ 3\ (inouts_v\ x)))\ 0]))] =$
 $inouts_v'\ x)$

```

assume  $a2: 0 < x$ 
assume  $a3: 1 \leq inouts_v\ x!(2) * 2$ 
from  $a1$  have  $11: \forall x. length(inouts_v\ x) = 4$ 
  using  $a2$  by blast
have  $12: hd(drop\ 3\ (inouts_v\ x)) = (inouts_v\ x!(3))$ 
  using  $11$  by (simp add: hd-drop-conv-nth)
have  $13: (hd (take\ 3\ (inouts_v\ (x - Suc\ 0)))) = (hd (inouts_v\ (x - Suc\ 0)))$ 
  using  $a1$  by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have  $14: (hd (take\ 3\ (inouts_v\ (x - Suc\ 0)))) = (hd (inouts_v\ (x - Suc\ 0)))$ 
  using  $a1$  by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have  $15: (hd (tl (take\ 3\ (inouts_v\ (x - Suc\ 0)))) = (hd (tl (inouts_v\ (x - Suc\ 0))))$ 
  by (metis Zero-not-Suc append-take-drop-id hd-append2 numeral-3-eq-3 take-eq-Nil take-tl)
show  $[min (hd (inouts_v\ (x - Suc\ 0))) (hd (tl (inouts_v\ (x - Suc\ 0)))) + 1,$ 
 $min (hd (inouts_v\ (x - Suc\ 0))) (hd (tl (inouts_v\ (x - Suc\ 0)))) + 1,$ 
 $real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v\ x!(3))\ 0]))),$ 
 $real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v\ x!(3))\ 0]))] =$ 
 $inouts_v'\ x$ 

```



```

    using 11 12 13 14 15 by (metis a1 a2 a3)
next
fix okv and inoutsv::nat⇒real list and okv' and inoutsv'::nat⇒real list
and x::nat
assume a1: ∀x. (x = 0 →
  (1 ≤ inoutsv 0!(2) * 2 →
    length(inoutsv 0) = 4 ∧
    length(inoutsv' 0) = 4 ∧
    [1, 1, real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv 0)))
0]))),
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv 0))) 0])))] =
inoutsv' 0) ∧
  (¬ 1 ≤ inoutsv 0!(2) * 2 →
    length(inoutsv 0) = 4 ∧
    length(inoutsv' 0) = 4 ∧
    [0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv 0)))
0]))),
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv 0))) 0])))] =
inoutsv' 0) ∧
  (0 < x →
    (1 ≤ inoutsv x!(2) * 2 →
      length(inoutsv x) = 4 ∧
      length(inoutsv' x) = 4 ∧
      [min (hd (take 3 (inoutsv (x - Suc 0)))) (hd (tl (take 3 (inoutsv (x - Suc 0)))))) + 1,
        min (hd (take 3 (inoutsv (x - Suc 0)))) (hd (tl (take 3 (inoutsv (x - Suc 0)))))) + 1,
        real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x))) 0]))),
        real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x))) 0])))] =
inoutsv' x) ∧
      (¬ 1 ≤ inoutsv x!(2) * 2 →
        length(inoutsv x) = 4 ∧
        length(inoutsv' x) = 4 ∧
        [0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x)))
0]))),
          real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x))) 0])))] =
inoutsv' x)
    ]
  )
have 11: hd (drop 3 (inoutsv x)) = inoutsv x!(3)
  by (metis a1 eval-nat-numeral(2) gr-zeroI hd-drop-conv-nth lessI semiring-norm(26)
semiring-norm(27))
show ¬ 1 ≤ inoutsv x!(2) * 2 →
  length(inoutsv x) = 4 ∧
  length(inoutsv' x) = 4 ∧
  [0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!(3)) 0]))),
    real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!(3)) 0])))] =
inoutsv' x
  apply (auto)
  using a1 gr-zeroI apply blast
  using a1 gr-zeroI apply blast
  by (metis 11 a1 neq0-conv)
next
show ∧ okv inoutsv okv' inoutsv'.
  okv →
  okv' ∧
  (∀x. (x = 0 →
    (1 ≤ inoutsv 0!2 * 2 →
      length(inoutsv 0) = 4 ∧

```

$$\begin{aligned}
& \text{length}(\text{inouts}_v' 0) = 4 \wedge \\
& [1, 1, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v 0!3) 0])), \\
& \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v 0!3) 0])))] = \\
& \text{inouts}_v' 0) \wedge \\
& (\neg 1 \leq \text{inouts}_v 0!2 * 2 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v 0) = 4 \wedge \\
& \quad \text{length}(\text{inouts}_v' 0) = 4 \wedge \\
& \quad [0, 0, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v 0!3) 0])), \\
& \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v 0!3) 0])))] = \\
& \quad \text{inouts}_v' 0)) \wedge \\
& (0 < x \longrightarrow \\
& \quad (1 \leq \text{inouts}_v x!2 * 2 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v x) = 4 \wedge \\
& \quad \quad \text{length}(\text{inouts}_v' x) = 4 \wedge \\
& \quad \quad [\min (\text{hd} (\text{inouts}_v (x - \text{Suc } 0))) (\text{hd} (\text{tl} (\text{inouts}_v (x - \text{Suc } 0)))) + 1, \\
& \quad \quad \min (\text{hd} (\text{inouts}_v (x - \text{Suc } 0))) (\text{hd} (\text{tl} (\text{inouts}_v (x - \text{Suc } 0)))) + 1, \\
& \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v x!3) 0])), \\
& \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v x!3) 0])))] = \\
& \quad \quad \text{inouts}_v' x) \wedge \\
& \quad (\neg 1 \leq \text{inouts}_v x!2 * 2 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v x) = 4 \wedge \\
& \quad \quad \text{length}(\text{inouts}_v' x) = 4 \wedge \\
& \quad \quad [0, 0, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v x!3) 0])), \\
& \quad \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v x!3) 0])))] = \\
& \quad \quad \text{inouts}_v' x)) \implies \\
& \text{ok}_v \longrightarrow \\
& \text{ok}_v' \wedge \\
& (\forall x. (x = 0 \longrightarrow \\
& \quad (1 \leq \text{inouts}_v 0!2 * 2 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v 0) = 4 \wedge \\
& \quad \quad \text{length}(\text{inouts}_v' 0) = 4 \wedge \\
& \quad [1, 1, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v 0))) \\
& \quad \quad \quad 0])), \\
& \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v 0))) \\
& \quad \quad \quad 0])))] \\
& \quad \quad \quad \text{inouts}_v' 0) \wedge \\
& \quad (\neg 1 \leq \text{inouts}_v 0!2 * 2 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v 0) = 4 \wedge \\
& \quad \quad \text{length}(\text{inouts}_v' 0) = 4 \wedge \\
& \quad [0, 0, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v 0))) \\
& \quad \quad \quad 0])), \\
& \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v 0))) \\
& \quad \quad \quad 0])))] \\
& \quad \quad \quad \text{inouts}_v' 0)) \wedge \\
& \quad (0 < x \longrightarrow \\
& \quad \quad (1 \leq \text{inouts}_v x!2 * 2 \longrightarrow \\
& \quad \quad \quad \text{length}(\text{inouts}_v x) = 4 \wedge \\
& \quad \quad \quad \text{length}(\text{inouts}_v' x) = 4 \wedge \\
& \quad \quad \quad [\min (\text{hd} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) (\text{hd} (\text{tl} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) + 1, \\
& \quad \quad \quad \min (\text{hd} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) (\text{hd} (\text{tl} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) + 1, \\
& \quad \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v x))) 0])), \\
& \quad \quad \quad \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{hd} (\text{drop } 3 (\text{inouts}_v x))) 0])))] \\
& \quad \quad \quad \text{inouts}_v' x) \wedge \\
& \quad \quad (\neg 1 \leq \text{inouts}_v x!2 * 2 \longrightarrow
\end{aligned}$$

```

length(inouts_v x) = 4 ∧
length(inouts_v' x) = 4 ∧
[0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inouts_v x)))
0])))),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inouts_v x))) 0]))))]
=
inouts_v' x))
apply (clarify)
apply (rule conjI)
apply (clarify)
apply (rule conjI)
apply (clarify)
apply (rule conjI)
apply blast
apply (rule conjI)
apply blast
apply (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
apply (clarify)
apply (rule conjI)
apply blast
apply (rule conjI)
apply blast
apply (metis eval-nat-numeral(2) hd-drop-conv-nth lessI semiring-norm(26) semiring-norm(27))
apply (clarify)
apply (rule conjI)
apply (clarify)
apply (rule conjI)
apply blast
apply (rule conjI)
apply blast
proof -
fix ok_v and inouts_v::nat⇒real list and ok_v' and inouts_v'::nat⇒real list
and x::nat
assume a1: ∀ x. (x = 0 →
(1 ≤ inouts_v 0!2 * 2 →
length(inouts_v 0) = 4 ∧
length(inouts_v' 0) = 4 ∧
[1, 1, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!3) 0]))),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!3) 0])))] =
inouts_v' 0) ∧
(¬ 1 ≤ inouts_v 0!2 * 2 →
length(inouts_v 0) = 4 ∧
length(inouts_v' 0) = 4 ∧
[0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!3) 0]))),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!3) 0])))] =
inouts_v' 0)) ∧
(0 < x →
(1 ≤ inouts_v x!2 * 2 →
length(inouts_v x) = 4 ∧
length(inouts_v' x) = 4 ∧
[min (hd (inouts_v (x - Suc 0))) (hd (tl (inouts_v (x - Suc 0)))) + 1,
min (hd (inouts_v (x - Suc 0))) (hd (tl (inouts_v (x - Suc 0)))) + 1,
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!3) 0]))),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!3) 0])))] =
inouts_v' x) ∧

```

```

(¬ 1 ≤ inoutsv x!2 * 2 →
length(inoutsv x) = 4 ∧
length(inoutsv' x) = 4 ∧
[0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!3) 0])),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!3) 0])))] =
inoutsv' x)
assume a2: 0 < x
assume a3: 1 ≤ inoutsv x!(2) * 2
from a1 have 11: ∀ x. length(inoutsv x) = 4
  using a2 by blast
have 12: hd(drop 3 (inoutsv x)) = (inoutsv x!(3))
  using 11 by (simp add: hd-drop-conv-nth)
have 13: (hd (take 3 (inoutsv (x - Suc 0)))) = (hd (inoutsv (x - Suc 0)))
  using a1 by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have 14: (hd (take 3 (inoutsv (x - Suc 0)))) = (hd (inoutsv (x - Suc 0)))
  using a1 by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have 15: (hd (tl (take 3 (inoutsv (x - Suc 0)))) = (hd (tl (inoutsv (x - Suc 0))))
by (metis Zero-not-Suc append-take-drop-id hd-append2 numeral-3-eq-3 take-eq-Nil take-tl)
show [min (hd (take 3 (inoutsv (x - Suc 0)))) (hd (tl (take 3 (inoutsv (x - Suc 0)))))]
+ 1,
      min (hd (take 3 (inoutsv (x - Suc 0)))) (hd (tl (take 3 (inoutsv (x - Suc 0)))) + 1,
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x)) 0])),
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (drop 3 (inoutsv x))
0])))] =
      inoutsv' x
  using 11 12 13 14 15 by (metis a1 a2 a3)
next
fix okv and inoutsv::nat⇒real list and okv' and inoutsv'::nat⇒real list
  and x::nat
assume a1: ∀ x. (x = 0 →
(1 ≤ inoutsv 0!2 * 2 →
length(inoutsv 0) = 4 ∧
length(inoutsv' 0) = 4 ∧
[1, 1, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!3) 0])),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!3) 0])))] =
inoutsv' 0) ∧
(¬ 1 ≤ inoutsv 0!2 * 2 →
length(inoutsv 0) = 4 ∧
length(inoutsv' 0) = 4 ∧
[0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!3) 0])),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!3) 0])))] =
inoutsv' 0)) ∧
(0 < x →
(1 ≤ inoutsv x!2 * 2 →
length(inoutsv x) = 4 ∧
length(inoutsv' x) = 4 ∧
[min (hd (inoutsv (x - Suc 0))) (hd (tl (inoutsv (x - Suc 0)))) + 1,
min (hd (inoutsv (x - Suc 0))) (hd (tl (inoutsv (x - Suc 0)))) + 1,
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!3) 0])),
real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!3) 0])))] =
inoutsv' x) ∧
(¬ 1 ≤ inoutsv x!2 * 2 →
length(inoutsv x) = 4 ∧
length(inoutsv' x) = 4 ∧
[0, 0, real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!3) 0])),

```

```

    real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!3) 0]))) =
    inouts_v' x)
assume a2: 0 < x
from a1 have 11:  $\forall x. \text{length}(\text{inouts}_v x) = 4$ 
  using a2 by blast
have 12:  $\text{hd}(\text{drop } 3 (\text{inouts}_v x)) = (\text{inouts}_v x!(3))$ 
  using 11 by (simp add: hd-drop-conv-nth)
have 13:  $(\text{hd} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) = (\text{hd} (\text{inouts}_v (x - \text{Suc } 0)))$ 
  using a1 by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have 14:  $(\text{hd} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) = (\text{hd} (\text{inouts}_v (x - \text{Suc } 0)))$ 
  using a1 by (metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral)
have 15:  $(\text{hd} (\text{tl} (\text{take } 3 (\text{inouts}_v (x - \text{Suc } 0)))) = (\text{hd} (\text{tl} (\text{inouts}_v (x - \text{Suc } 0))))$ 
by (metis Zero-not-Suc append-take-drop-id hd-append2 numeral-3-eq-3 take-eq-Nil take-tl)
show  $\neg 1 \leq \text{inouts}_v x!(2) * 2 \longrightarrow$ 
   $\text{length}(\text{inouts}_v x) = 4 \wedge$ 
   $\text{length}(\text{inouts}_v' x) = 4 \wedge$ 
   $[0, 0, \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{hd} (\text{drop } 3 (\text{inouts}_v x))$ 
0])))],
```

=

```

    inouts_v' x
apply (clarify)
apply (rule conjI)
apply (simp add: 11)
apply (rule conjI)
using a1 a2 apply blast
using 11 12 13 14 15
by (simp add: a1 a2)
qed
qed
qed
have simblock-f1: SimBlock 4 4 (FBlock ( $\lambda x n. \text{True}$ ) (4) 4
  (( $\lambda x n.$ 
    [if (x n)!2  $\geq$  0.5
      then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
      if (x n)!2  $\geq$  0.5
      then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
      real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3) 0)]))),
      real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3) 0)])))]))
using simblock-variableTimer1 simblock-variableTimer2
by (metis (no-types, lifting) One-nat-def SimBlock-FBlock-parallel-comp Suc-eq-plus1
  eval-nat-numeral(2) f1-0 f1-1 numeral-code(2) semiring-norm(26) semiring-norm(27))
have inps-f1: inps (FBlock ( $\lambda x n. \text{True}$ ) (4) 4
  (( $\lambda x n.$ 
    [if (x n)!2  $\geq$  0.5
      then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
      if (x n)!2  $\geq$  0.5
      then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,
      real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3) 0)]))),
      real-of-int (int32 (RoundZero(real-of-int [Rate * (max ((x n)!3) 0)])))])) = 4
using simblock-f1 using inps-P by blast
have outps-f1: outps (FBlock ( $\lambda x n. \text{True}$ ) (4) 4
  (( $\lambda x n.$ 
    [if (x n)!2  $\geq$  0.5
      then ((if n = 0 then 0 else (min (hd(x (n-1))) (hd(tl(x (n-1))))) + 1) else 0,

```

$if (x\ n)!2 \geq 0.5$
 $then ((if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1))))) + 1)\ else\ 0,$
 $real-of-int\ (int32\ (RoundZero(real-of-int\ [Rate * (max\ ((x\ n)!3)\ 0])))$,
 $real-of-int\ (int32\ (RoundZero(real-of-int\ [Rate * (max\ ((x\ n)!3)\ 0])))) = 4$
using *simblock-f1* **using** *outps-P* **by** *blast*

let $?f2-f = ((\lambda x\ n.$
 $[if\ (x\ n)!2 \geq 0.5$
 $then\ ((if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1))))) + 1)\ else\ 0,$
 $if\ (x\ n)!2 \geq 0.5$
 $then\ ((if\ n = 0\ then\ 0\ else\ (min\ (hd(x\ (n-1)))\ (hd(tl(x\ (n-1))))) + 1)\ else\ 0,$
 $real-of-int\ (int32\ (RoundZero(real-of-int\ [Rate * (max\ ((x\ n)!3)\ 0])))$,
 $real-of-int\ (int32\ (RoundZero(real-of-int\ [Rate * (max\ ((x\ n)!3)\ 0]))))])$
let $?f2 = (FBlock\ (\lambda x\ n.\ True)\ (4)\ 4\ ?f2-f)$
let $?f2-xx = (\lambda(inouts_0::nat \Rightarrow real\ list).\ \lambda na.\ vT-fd-sol-1$
 $(\lambda n1.\ hd(inouts_0\ n1))\ (\lambda n1.\ (inouts_0\ n1)!1)\ na)$
have $f2: ((variableTimer1\ ||_B\ variableTimer2)\ f_D\ (0,0))$
 $= ?f2\ f_D\ (0,0)$
using $f1\ f1-0\ f1-1$ **by** *auto*
have $is-solution-f2: is-Solution\ 0\ 0\ 4\ 4\ ?f2-f\ ?f2-xx$
apply (*simp* *add: is-Solution-def*)
apply (*rule allI*)
apply (*simp* *add: f-PreFD-def*)
apply (*clarify*)
using $vT-fd-sol-1-is-a-solution$ **by** *blast*
have $unique-f2: Solvable-unique\ 0\ 0\ 4\ 4\ ?f2-f$
apply (*simp* *add: Solvable-unique-def*)
apply (*rule allI, clarify, simp* *add: f-PreFD-def*)
apply (*rule ex-ex1I*)
apply (*rule-tac* $x = \lambda na.\ vT-fd-sol-1$
 $(\lambda n1.\ hd(inouts_0\ n1))\ (\lambda n1.\ (inouts_0\ n1)!1)\ na$ **in** *exI*)
apply (*simp*)
apply (*rule allI*)
using $vT-fd-sol-1-is-a-solution$ **apply** (*simp*)
proof –
fix $inouts_0::nat \Rightarrow real\ list$ **and** $xx\ y ::nat \Rightarrow real$
assume $a1: \forall x.\ length(inouts_0\ x) = 3$
assume $a2: \forall n.\ (n = 0 \longrightarrow (1 \leq inouts_0\ 0!(Suc\ 0) * 2 \longrightarrow xx\ 0 = 1) \wedge$
 $(\neg\ 1 \leq inouts_0\ 0!(Suc\ 0) * 2 \longrightarrow xx\ 0 = 0)) \wedge$
 $(0 < n \longrightarrow$
 $(1 \leq inouts_0\ n!(Suc\ 0) * 2 \longrightarrow xx\ n = min\ (xx\ (n - Suc\ 0))\ (hd\ (inouts_0\ (n - Suc\ 0))) +$
 $1) \wedge$
 $(\neg\ 1 \leq inouts_0\ n!(Suc\ 0) * 2 \longrightarrow xx\ n = 0))$
assume $a3: \forall n.\ (n = 0 \longrightarrow (1 \leq inouts_0\ 0!(Suc\ 0) * 2 \longrightarrow y\ 0 = 1) \wedge$
 $(\neg\ 1 \leq inouts_0\ 0!(Suc\ 0) * 2 \longrightarrow y\ 0 = 0)) \wedge$
 $(0 < n \longrightarrow$
 $(1 \leq inouts_0\ n!(Suc\ 0) * 2 \longrightarrow y\ n = min\ (y\ (n - Suc\ 0))\ (hd\ (inouts_0\ (n - Suc\ 0))) +$
 $1) \wedge$
 $(\neg\ 1 \leq inouts_0\ n!(Suc\ 0) * 2 \longrightarrow y\ n = 0))$
have $1: \forall n.\ xx\ n = y\ n$
apply (*rule allI*)
proof –
fix $n::nat$
show $xx\ n = y\ n$
proof (*induct* n)

```

    case 0
    then show ?case
      using a2 a3 by metis
    next
    case (Suc n) note IH = this
    then show ?case
      using a2 a3 by (metis One-nat-def diff-Suc-1 zero-less-Suc)
  qed
  qed
  show  $xx = y$ 
  by (simp add: 1 fun-eq)
  qed
  let ?f3-f = ( $\lambda x na. [if 1 \leq x na!(Suc 0) * 2$ 
    then ( $if na = 0$  then 0
      else  $min ((vT-fd-sol-1 (\lambda n1. hd (x n1)) (\lambda n1. x n1!(Suc 0))) (na - 1))$ 
        ( $hd (x (na - 1)))) + 1$ 
      else 0,
    real-of-int (int32 (RoundZero (real-of-int [ $Rate * max (x na!(2)) 0$ ]))),
    real-of-int (int32 (RoundZero (real-of-int [ $Rate * max (x na!(2)) 0$ ])))]))
  have f2-0:
    ?f2 fD (0,0) =
      (FBlock ( $\lambda x n. True$ ) (4-1) (4-1)
        ( $\lambda x na. ((f-PostFD 0)$ 
          o ?f2-f
          o ( $f-PreFD$  (?f2-xx x) 0)) x na))
    using is-solution-f2 unique-f2 simblock-f1 FBlock-feedback' by blast
  then have f2-1:
    ... = FBlock ( $\lambda x n. True$ ) 3 3 ?f3-f
    apply (simp (no-asm) add: f-PreFD-def f-PostFD-def)
    using f-PreFD-def
    by (metis (lifting) append.left-neutral drop-0 f-PreFD-def list.sel(1) list.sel(3) take-0)
  have simblock-f2-0: SimBlock (4-1) (4-1) (?f2 fD (0,0))
    using simblock-f1 unique-f2 Solvable-unique-is-solvable SimBlock-FBlock-feedback by blast
  then have simblock-f2: SimBlock 3 3 (FBlock ( $\lambda x n. True$ ) 3 3 ?f3-f)
    by (metis (no-types, lifting) Suc-eq-plus1 add-diff-cancel-right' eval-nat-numeral(2) f2-0
      f2-1 semiring-norm(26) semiring-norm(27))
  have inps-f2: inps (FBlock ( $\lambda x n. True$ ) 3 3 ?f3-f) = 3
    using simblock-f2 using inps-P by blast
  have outps-f2: outps (FBlock ( $\lambda x n. True$ ) 3 3 ?f3-f) = 3
    using simblock-f2 using outps-P by blast

  have f3: (((variableTimer1  $\parallel_B$  variableTimer2) fD (0,0)) fD (0,2))
    = (FBlock ( $\lambda x n. True$ ) 3 3 ?f3-f) fD (0,2)
    using f2 f2-0 f2-1 by auto
  let ?f3-xx = ( $\lambda (inouts_0::nat \Rightarrow real list). \lambda na.$ 
    real-of-int (int32 (RoundZero (real-of-int [ $Rate * max (inouts_0 na!(1)) 0$ ]))))
  have is-solution-f3: is-Solution 0 2 3 3 ?f3-f ?f3-xx
    apply (simp add: is-Solution-def)
    apply (rule allI)
    by (simp add: f-PreFD-def)
  have unique-f3: Solvable-unique 0 2 3 3 ?f3-f
    apply (simp add: Solvable-unique-def)
    apply (rule allI, clarify, simp add: f-PreFD-def)
    apply (rule ex-ex1I)
    apply (rule-tac x =  $\lambda na.$ 

```

```

    real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts0 na!(1)) 0]))) in exI)
  apply (simp)
  by (simp add: ext)
  have simp-1: ∀ x na. (λx na. [if 1 ≤ x na!(0) * 2
    then (if na = 0 then 0
      else min (vT-fd-sol-1
        (λn1. hd (f-PreFD
          (λna. real-of-int
            (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
            0 x n1))
        (λn1. f-PreFD
          (λna. real-of-int
            (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
            0 x n1!(Suc 0))
          (na - 1))
        (hd (f-PreFD
          (λna. real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0))
0])))
            0 x (na - 1)))))) +
      1
    else 0,
    real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) x na
  = (λx na. [if 1 ≤ x na!(0) * 2
    then (if na = 0 then 0
      else min (vT-fd-sol-1
        (λn1. (λna. real-of-int
          (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
        (λn1. (x n1)!(0)) (na - 1))
        ((λna. real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
          (na - 1))) + 1
      else 0,
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) x na
  by (simp add: f-PreFD-def)
  let ?f4-f = (λx na. [if 1 ≤ x na!(0) * 2
    then (if na = 0 then 0
      else min (vT-fd-sol-1
        (λn1. (λna. real-of-int
          (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
        (λn1. (x n1)!(0)) (na - 1))
        ((λna. real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
          (na - 1))) + 1
      else 0,
      real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
  have f3-0: (FBlock (λx n. True) 3 3 ?f3-f) fD (0,2)
    = (FBlock (λx n. True) (3-1) (3-1)
      (λx na. ((f-PostFD 2)
        o ?f3-f
        o (f-PreFD (?f3-xx x) 0)) x na))
  using is-solution-f3 unique-f3 simblock-f2 FBlock-feedback' by blast
  then have f3-1: ... = FBlock (λx n. True) 2 2 ?f4-f
  apply (simp (no-asm) add: f-PreFD-def f-PostFD-def)
  by (simp add: simp-1)
  have simblock-f3-0: SimBlock (3-1) (3-1) ((FBlock (λx n. True) 3 3 ?f3-f) fD (0,2))
    using simblock-f2 unique-f3 Solvable-unique-is-solvable SimBlock-FBlock-feedback by blast
  then have simblock-f3: SimBlock 2 2 (FBlock (λx n. True) 2 2 ?f4-f)

```



```

by (metis (no-types, lifting) One-nat-def Suc-1 diff-Suc-1 f3-0 f3-1 numeral-3-eq-3)

have simp-f4:  $\forall x n. (f\text{-RopGT} \circ ?f4\text{-f}) x n = ?vt\text{-f} x n$ 
  using f-RopGT-def by simp
have f4: variableTimer = (FBlock ( $\lambda x n. \text{True}$ ) 2 2 ?f4-f) ;; RopGT
  using f3 f3-0 f3-1 variableTimer-def by auto
then have f4-0: ... = FBlock ( $\lambda x n. \text{True}$ ) 2 1 (f-RopGT  $\circ$  ?f4-f)
  using simblock-f3 SimBlock-RopGT FBlock-seq-comp by (simp add: RopGT-def)
then have f4-1: ... = FBlock ( $\lambda x n. \text{True}$ ) 2 1 ?vt-f
  using simp-f4 by presburger
show ?thesis
  using f4 f4-0 f4-1 by auto
qed

```

C.1.1 Verification

vt-req-00: if *door_open* is false (door is closed), then the output of this subsystem is false. This is not a requirement described in the paper but we believe it should hold for this subsystem.

Current Simulink diagram cannot guarantee this property because the type conversion `int32` could cause its output less than 0 (i.e. $4294967295 = -10$), finally the output of *variableTimer* could be true. It violates our requirement. In the original Simulink block diagram, this *variableTimer* is a subsystem of *post-landing-finalize* which itself is a subsystem of aircraft cabin pressure and environment control system applications. Therefore, its second input (*door_open_time*) relies on the outputs of other subsystem (Timing Computation), and *variableTimer* actually makes assumptions on its input.

However, taking *variableTimer* alone, we try to verify this property either strengthen its precondition on the input (*door_open_times* is always larger or equal to 0 and less than $2147483647/\text{Rate}$), or change `int32` to `uint32` for the type conversion block, or change the data type of this input to unsigned integer.

In the lemma below, we proved this property holds if we make an assumption on its values.

lemma *vt-req-00*:

```

(( $\forall n::\text{nat} \cdot$ 
  ( $\langle\langle \lambda x n. (\text{hd}(x n) = 0 \vee \text{hd}(x n) = 1) \wedge (* \text{ the first input door-open is boolean. } *)$ 
    ( $\text{hd}(\text{tl}(x n)) \geq 0 \wedge \text{hd}(\text{tl}(x n)) < 214748364\rangle\rangle$ )
    ( $\&\text{inouts}$ )a ( $\langle n \rangle$ )a::sim-state upred)
   $\vdash_n$ 
  ( $\forall n::\text{nat} \cdot$ 
    ( $(\#_u(\$inouts (\langle n \rangle)_a)) =_u \langle 2 \rangle$ )  $\wedge$ 
    ( $(\#_u(\$inouts' (\langle n \rangle)_a)) =_u \langle 1 \rangle$ )  $\wedge$ 
    ( $\text{head}_u((\$inouts' (\langle n \rangle)_a)) =_u 0 \Rightarrow (\text{head}_u((\$inouts (\langle n \rangle)_a)) =_u 0)$ )
  ))  $\sqsubseteq$  variableTimer

```

apply (*simp* (*no-asm*) *add: variableTimer-simp*)

apply (*simp* *add: FBlock-def*)

apply (*rel-simp*)

proof –

fix $ok_v::\text{bool}$ **and** $inouts_v::\text{nat} \Rightarrow \text{real list}$ **and** $ok_v'::\text{bool}$ **and** $inouts_v'::\text{nat} \Rightarrow \text{real list}$

and $x::\text{nat}$

assume $a1: \forall x. (\text{hd}(inouts_v x) = 0 \vee \text{hd}(inouts_v x) = 1) \wedge$

$(0 \leq \text{hd}(\text{tl}(inouts_v x)) \wedge \text{hd}(\text{tl}(inouts_v x)) < 214748364)$

assume $a2: \text{hd}(inouts_v x) = 0$

assume $a3: \forall x. (x = 0 \longrightarrow$

$(1 \leq inouts_v 0!(0) * 2 \longrightarrow$

$(\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)) < 1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ 0) = 2 \wedge \text{length}(\text{inouts}_v' \ 0) = \text{Suc } 0 \wedge [1] = \text{inouts}_v' \ 0 \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)) < 1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ 0) = 2 \wedge \text{length}(\text{inouts}_v' \ 0) = \text{Suc } 0 \wedge [0] = \text{inouts}_v' \ 0) \wedge$
 $(\neg 1 \leq \text{inouts}_v \ 0!(0) * 2 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ 0) = 2 \wedge \text{length}(\text{inouts}_v' \ 0) = \text{Suc } 0 \wedge [1] = \text{inouts}_v' \ 0 \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ 0) = 2 \wedge \text{length}(\text{inouts}_v' \ 0) = \text{Suc } 0 \wedge [0] = \text{inouts}_v' \ 0))) \wedge$
 $(0 < x \longrightarrow$
 $(1 \leq \text{inouts}_v \ x!(0) * 2 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))$
 $< \min (\text{vT-fd-sol-1}$
 $\quad (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0))$
 $0 \rceil))))))$
 $\quad (\lambda n1. \text{inouts}_v \ n1!(0)) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0 \rceil)))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 2 \wedge \text{length}(\text{inouts}_v' \ x) = \text{Suc } 0 \wedge [1] = \text{inouts}_v' \ x \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))$
 $< \min (\text{vT-fd-sol-1}$
 $\quad (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0))$
 $0 \rceil))))))$
 $\quad (\lambda n1. \text{inouts}_v \ n1!(0)) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0 \rceil)))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 2 \wedge \text{length}(\text{inouts}_v' \ x) = \text{Suc } 0 \wedge [0] = \text{inouts}_v' \ x) \wedge$
 $(\neg 1 \leq \text{inouts}_v \ x!(0) * 2 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 2 \wedge \text{length}(\text{inouts}_v' \ x) = \text{Suc } 0 \wedge [1] = \text{inouts}_v' \ x \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 2 \wedge \text{length}(\text{inouts}_v' \ x) = \text{Suc } 0 \wedge [0] = \text{inouts}_v' \ x)))$
have 1: $\forall x. \text{length}(\text{inouts}_v \ x) = 2$
using *a3 neq0-conv by blast*
have 2: $\text{inouts}_v \ x!(0) = 0$
using *1 a2 by (metis hd-conv-nth list.size(3) zero-not-eq-two)*
have 3: $\forall x. (0 \leq \text{inouts}_v \ x!(\text{Suc } 0) \wedge \text{inouts}_v \ x!(\text{Suc } 0) < 214748364)$
using *a1*
by *(metis 1 One-nat-def diff-Suc-1 hd-conv-nth length-greater-0-conv length-tl less-numeral-extra(1) nth-tl numeral-2-eq-2)*
have 30: $\forall x. \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 < \text{Rate} * 214748364 \wedge$
 $\text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \geq 0$
using *3 by simp*
have $\forall x. \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil < (\text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 + 1)$
using *ceiling-correct by linarith*
then have $\forall x. \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil < (\text{Rate} * 214748364 + 1)$
using *30 by (metis add.commute cancel-ab-semigroup-add-class.add-diff-cancel-left' ceiling-less-iff less-eq-real-def numeral-times-numeral-of-int-numeral one-plus-numeral)*
then have 31: $\forall x. \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil < (\text{Rate} * 214748364 + 1) \wedge$
 $\lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil \geq 0$
using *30 by (smt ceiling-le-zero ceiling-zero)*
have 32: $\forall x. \text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil < (\text{Rate} * 214748364 + 1) \wedge$
 $\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil \geq 0$

using 31 **by** (*simp*)
have 33: $\forall x. \text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)$
 $= \lfloor \text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil \rfloor$
using *RoundZero-def* **by** (*simp*)
have 34: $\forall x. \text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil) < (\text{Rate} * 214748364 +$
1) \wedge
 $\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil) \geq 0$
using 33 31 **by** *auto*
have 35: $\forall x. \text{int32} (\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))$
 $= \text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)$
using 34 *int32-eq* **by** *smt*
have 36: $\forall x. \text{int32} (\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))$
 $< (\text{Rate} * 214748364 + 1) \wedge$
 $\text{int32} (\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) \geq 0$
using 35 34 **by** (*simp*)
show *hd* (*inouts*_v' *x*) = 0
using *a2 a3 36 2*
by (*metis* (*no-types*, *lifting*) *less-numeral-extra*(1) *list.sel*(1) *mult-zero-left neq0-conv not-le*)
qed

lemma *door-open-time-range*:

fixes *x* :: *real* **and** *door-open-time*::*real*
assumes *door-open-time* < 214748364 \wedge *door-open-time* > 0
assumes (0 \leq *x* \wedge *x* < *door-open-time*)
shows *int32* (*RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$)) \geq 0 \wedge
 $\text{int32} (\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max x 0 \rceil)) < (\text{Rate} * \text{door-open-time} + 1)$
proof –
have 0: $\text{Rate} * \max x 0 < \text{Rate} * \text{door-open-time} \wedge \text{Rate} * \max x 0 \geq 0$
using *assms* **by** *simp*
have 1: $\lceil \text{Rate} * \max x 0 \rceil < (\text{Rate} * \max x 0 + 1)$
using *ceiling-correct* **by** *linarith*
then have $\lceil \text{Rate} * \max x 0 \rceil < (\text{Rate} * \text{door-open-time} + 1)$
using 0 *assms* **by** *linarith*
then have 2: $\lceil \text{Rate} * \max x 0 \rceil < (\text{Rate} * \text{door-open-time} + 1) \wedge$
 $\lceil \text{Rate} * \max x 0 \rceil \geq 0$
using 0 **by** (*smt ceiling-le-zero ceiling-zero*)
have 3: *real-of-int* $\lceil \text{Rate} * \max x 0 \rceil < (\text{Rate} * \text{door-open-time} + 1) \wedge$
real-of-int $\lceil \text{Rate} * \max x 0 \rceil \geq 0$
using 2 **by** (*simp*)
have 4: *RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$)
 $= \lfloor \text{real-of-int } \lceil \text{Rate} * \max x 0 \rceil \rfloor$
using *RoundZero-def* **by** (*simp*)
have 5: *RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$) < (*Rate* * *door-open-time* + 1) \wedge
RoundZero (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$) \geq 0
using 3 4 **by** *auto*
have 51: *RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$) < (*Rate* * 214748364 + 1) \wedge
RoundZero (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$) \geq 0
using 5 *assms* **by** *auto*
have 6: *int32* (*RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$))
 $= \text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max x 0 \rceil)$
using 51 *int32-eq* *assms* **by** *simp*
have 7: *int32* (*RoundZero* (*real-of-int* $\lceil \text{Rate} * \max x 0 \rceil$))
 $< (\text{Rate} * \text{door-open-time} + 1) \wedge$
 $\text{int32} (\text{RoundZero} (\text{real-of-int } \lceil \text{Rate} * \max x 0 \rceil)) \geq 0$
using 5 6 **by** (*simp*)

```

show ?thesis
  using 7 by blast
qed

```

C.2 Subsystem: *rise1Shot*

The *rise1Shot* subsystem is used for the purpose of making sure the finalize event is only triggered by once if doors are continuously open.

definition *rise1Shot* \equiv

$$(Split2 \ ; \ ; \ (Id \parallel_B \ (UnitDelay \ 1.0 \ (*3*)) \ ; \ ; \ LopNOT \ (*4*)) \ ; \ ; \ LopAND \ 2 \ (*Rise-1*))$$

rise1Shot-simp-pat-f gives the function definition of the finally simplified subsystem.

abbreviation *rise1Shot-simp-pat-f* $\equiv (\lambda x \ n. [if \ (hd(x \ n) \neq 0 \wedge (n > 0 \wedge hd(x \ (n-1)) = 0)) \ then \ 1 \ else \ 0])$

rise1Shot-simp-pat is the simplified block for the subsystem.

abbreviation *rise1Shot-simp-pat* $\equiv (FBlock \ (\lambda x \ n. \ True) \ 1 \ 1 \ rise1Shot-simp-pat-f)$

lemma *SimBlock-rise1Shot-simp*:

SimBlock 1 1 rise1Shot-simp-pat

apply (*rule SimBlock-FBlock*)

apply (*rule-tac x = λna. [0] in exI*)

apply (*rule-tac x = λna. [0] in exI*)

apply (*simp*)

by *simp*

rise1Shot-simp simplifies the subsystem into a block.

lemma *rise1Shot-simp*:

rise1Shot = rise1Shot-simp-pat

proof –

have *f1*: (*UnitDelay 1.0 (*3*) ; ; LopNOT (*4*)*) = *FBlock (λx n. True) 1 1 (f-LopNOT ∘ f-UnitDelay 1)*

using *SimBlock-LopNOT SimBlock-UnitDelay* **by** (*simp add: FBlock-seq-comp f-sim-blocks*)

have *simblock-f1*: *SimBlock 1 1 (FBlock (λx n. True) 1 1 (f-LopNOT ∘ f-UnitDelay 1))*

by (*metis (no-types, lifting) LopNOT-def SimBlock-LopNOT SimBlock-FBlock-seq-comp SimBlock-UnitDelay UnitDelay-def f1*)

have *f2*: (*Id ∥_B (UnitDelay 1.0 (*3*)) ; ; LopNOT (*4*))*)

= (*Id ∥_B FBlock (λx n. True) 1 1 (f-LopNOT ∘ f-UnitDelay 1)*)

using *f1* **by** (*simp*)

then have *f2-0*: ...

= (*FBlock (λx n. True) 2 2 (λx n. (((f-Id ∘ (λxx nn. take 1 (xx nn))) x n) • (((f-LopNOT ∘ f-UnitDelay 1) ∘ (λxx nn. drop 1 (xx nn)))) x n))*)

using *simblock-f1 SimBlock-Id FBlock-parallel-comp f1*

proof –

have $\bigwedge n \ na \ f. \neg \text{SimBlock } n \ na \ (FBlock \ (\lambda f \ n. \ True) \ n \ na \ f) \vee \text{FBlock } (\lambda f \ n. \ True) \ (n + 1) \ (na + 1) \ (\lambda fa \ na. \ (f \circ \ (\lambda f \ na. \ take \ n \ (f \ na))) \ fa \ na) \bullet \ (f-LopNOT \circ \ f-UnitDelay \ 1 \circ \ (\lambda f \ na. \ drop \ n \ (f \ na))) \ fa \ na) = \text{FBlock } (\lambda f \ n. \ True) \ n \ na \ f \parallel_B \text{FBlock } (\lambda f \ n. \ True) \ 1 \ 1 \ (f-LopNOT \circ \ f-UnitDelay \ 1)$

using *FBlock-parallel-comp simblock-f1* **by** *presburger*

then have $\neg \text{SimBlock } 1 \ 1 \ \text{simu-contract-real.Id} \vee \text{FBlock } (\lambda f \ n. \ True) \ (1 + 1) \ (1 + 1) \ (\lambda f \ n. \ (f-Id \circ \ (\lambda f \ n. \ take \ 1 \ (f \ n))) \ f \ n) \bullet \ (f-LopNOT \circ \ f-UnitDelay \ 1 \circ \ (\lambda f \ n. \ drop \ 1 \ (f \ n))) \ f \ n) = \text{FBlock } (\lambda f \ n. \ True) \ 1 \ 1 \ f-Id \parallel_B \text{FBlock } (\lambda f \ n. \ True) \ 1 \ 1 \ (f-LopNOT \circ \ f-UnitDelay \ 1)$

using *simu-contract-real.Id-def* **by** *presburger*

```

then show ?thesis
  by (metis (no-types) SimBlock-Id Suc-1 Suc-eq-plus1 simu-contract-real.Id-def)
qed
have simblock-f2: SimBlock 2 2
  (FBlock (λx n. True) 2 2 (λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)))
  by (metis (no-types, lifting) SimBlock-Id SimBlock-FBlock-parallel-comp Suc-1 Suc-eq-plus1
    f2-0 simblock-f1 simu-contract-real.Id-def)

have f3: Split2 ;; (Id ||B (UnitDelay 1.0 (*3*);; LopNOT (*4*)))
  = Split2 ;; (FBlock (λx n. True) 2 2 (λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)))
  using f2 f2-0 by (simp)
then have f3-0: ... = (FBlock (λx n. True) 1 2
  ((λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2))
  using SimBlock-Split2 simblock-f2 by (simp add: FBlock-seq-comp f-sim-blocks)
have simblock-f3: SimBlock 1 2 (FBlock (λx n. True) 1 2
  ((λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2))
  by (smt SimBlock-FBlock-seq-comp SimBlock-Split2 Split2-def f3-0 simblock-f2)

have f4: (Split2 ;; (Id ||B (UnitDelay 1.0 (*3*);; LopNOT (*4*))) ;; LopAND 2 (*Rise-1*))
  = (FBlock (λx n. True) 1 2
  ((λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2))
  ;; LopAND 2 (*Rise-1*)
  using f3 f3-0
by (smt LopAND-def FBlock-seq-comp SimBlock-LopAND SimBlock-FBlock-seq-comp SimBlock-Split2

  Split2-def comp-assoc f1 f2-0 neq0-conv simblock-f2 zero-not-eq-two)
have f4-0: ... = (FBlock (λx n. True) 1 1
  (f-LopAND o (λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
    (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2))
  using SimBlock-LopAND simblock-f3 by (simp add: LopAND-def FBlock-seq-comp comp-assoc)
have ∀ x n. (f-LopAND o (λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
  (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2) x n
  = ((λx n. [if (hd(x n) ≠ 0 ∧ (n > 0 ∧ hd(x (n-1)) = 0)) then 1 else 0]) x n
  using f-Id-def f-LopNOT-def f-UnitDelay-def f-LopAND-def f-Split2-def by simp)
then have (f-LopAND o (λx n. (((f-Id ◦ (λxx nn. take 1 (xx nn))) x n) •
  (((f-LopNOT ◦ f-UnitDelay 1) ◦ (λxx nn. drop 1 (xx nn)))) x n)) o f-Split2)
  = ((λx n. [if (hd(x n) ≠ 0 ∧ (n > 0 ∧ hd(x (n-1)) = 0)) then 1 else 0])
  by blast)
then have f4-1: (Split2 ;; (Id ||B (UnitDelay 1.0 (*3*);; LopNOT (*4*))) ;; LopAND 2
  (*Rise-1*)) =
  (FBlock (λx n. True) 1 1 (λx n. [if (hd(x n) ≠ 0 ∧ (n > 0 ∧ hd(x (n-1)) = 0)) then 1 else
  0]))
  using f4 f4-0 by (simp)
then show ?thesis
  by (simp add: rise1Shot-def)
qed

```

C.2.1 Verification

rise1shot-req-00 states that if the output of *rise1Shot* is true, then its present input must be true and the previous input must be false. In other word, the inputs that are continuously true won't trigger the output again.

lemma *rise1shot-req-00*:

```

((∀ n::nat · (
  «(λx n. (hd(x n) = 0 ∨ hd(x n) = 1))» (&inouts)a («n»)a::sim-state upred)
  ⊢n
  (∀ n::nat ·
    ((#u($inouts («n»)a)) =u «1») ∧
    ((#u($inouts' («n»)a)) =u «1») ∧
    (headu((($inouts' («n»)a)) =u 1) ⇒
      («n» >u 0 ∧ headu((($inouts («n»)a)) =u 1 ∧ headu((($inouts («n-1»)a)) =u 0))
    )) ⊆ rise1Shot
apply (simp (no-asm) add: rise1Shot-simp)
apply (simp add: FBlock-def)
apply (rel-simp)
by (metis list.sel(1) neq0-conv zero-neq-one)

```

C.3 Subsystem: Latch

This subsystem implements a SR AND-OR latch and it has two inputs: 1st is S (set) and 2nd is R (reset)

The first output is fed back into the first input.

definition *latch* ≡

```

((((UnitDelay 0 (*3*) ||B Id) ;; (LopOR 2 (*1*)))
  ||B
  (Id ;; LopNOT (*2*)))
  ) ;; (LopAND 2) (*Latch-1*) ;; Split2
  ) fD (0,0)

```

latch-rec-calc-output is the solution for the feedback.

fun *latch-rec-calc-output*:: (nat ⇒ real) ⇒ (nat ⇒ real) ⇒ nat ⇒ real **where**

```

latch-rec-calc-output S R 0 =
  (if R 0 = 0 then (if S 0 = 0 then 0 else 1.0) else 0) |
latch-rec-calc-output S R (Suc n) =
  (if R (Suc n) = 0 then (if S (Suc n) = 0 then (latch-rec-calc-output S R (n)) else 1.0) else 0)

```

lemma *latch-rec-calc-output-0-1*:

latch-rec-calc-output S R n = 0 ∨ latch-rec-calc-output S R n = 1

proof (induction n)

case 0

then show ?case **by** (simp)

next

case (Suc n)

then show ?case **by** (simp)

qed

lemma *latch-rec-calc-output-is-a-solution*:

fixes *inouts*₀::nat ⇒ real list **and** *n*::nat

assumes *a1*: ∀ x. length(*inouts*₀ x) = 2

shows $((0 < n \wedge \neg \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)))$
 $(\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (n - \text{Suc } 0) = 0 \vee \neg \text{hd } (\text{inouts}_0 \ n) = 0) \wedge$
 $\text{inouts}_0 \ n!(\text{Suc } 0) = 0 \longrightarrow$
 $\text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) \ n = 1) \wedge$
 $((n = 0 \vee \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)))$
 $(\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (n - \text{Suc } 0) = 0) \wedge \text{hd } (\text{inouts}_0 \ n) = 0 \longrightarrow$
 $\text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) \ n = 0) \wedge$
 $(\neg \text{inouts}_0 \ n!(\text{Suc } 0) = 0 \longrightarrow \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1))$
 $(\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) \ n = 0)$

apply (*rule conjI*)
apply (*clarify*)
proof –

assume $a2: 0 < n \wedge \neg \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (n$
 $- \text{Suc } 0) = 0 \vee$
 $\neg \text{hd } (\text{inouts}_0 \ n) = 0$
assume $a3: \text{inouts}_0 \ n!(\text{Suc } 0) = 0$
show $\text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) \ n = 1$
proof (*cases*)
assume $a4: 0 < n \wedge \neg \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (n$
 $- \text{Suc } 0) = 0$
from $a4$ **have** $1: n > 0$
by *blast*
have $11: \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) \ n =$
 $\text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (\text{Suc } (n - \text{Suc } 0))$
using 1 **by** *simp*
show *?thesis*
proof (*cases*)
assume $a5: \text{hd } (\text{inouts}_0 \ n) = 0$
from 11 **have** $12: \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (\text{Suc}$
 $(n - \text{Suc } 0))$
 $= \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (n - \text{Suc } 0)$
using $a3 \ a5$ **apply** (*simp (no-asm)*)
by (*simp add: 1*)
show *?thesis*
using $a4$ $\text{latch-rec-calc-output-0-1}$ **using** 12 **by** *auto*
next
assume $a5: \neg \text{hd } (\text{inouts}_0 \ n) = 0$
then have $12: \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0)) (\text{Suc } (n$
 $- \text{Suc } 0))$
 $= 1$
using $a3 \ a5$ **apply** (*simp (no-asm)*)
by (*simp add: 1*)
show *?thesis*
using $a4$ **using** 12 **by** *auto*
qed
next
assume $a4: \neg (0 < n \wedge \neg \text{latch-rec-calc-output } (\lambda n1. \text{hd } (\text{inouts}_0 \ n1)) (\lambda n1. \text{inouts}_0 \ n1!(\text{Suc } 0))$
 $(n - \text{Suc } 0) = 0)$
then have $1: \neg \text{hd } (\text{inouts}_0 \ n) = 0$
using $a2$ **by** *blast*
show *?thesis*
proof (*cases*)
assume $a5: n = 0$
show *?thesis*
using $a5$ **apply** (*simp*)

```

    using 1 a3 by blast
  next
    assume a5: ¬n = 0
    then have a5': n > 0
      by simp
    have 11: latch-rec-calc-output (λn1. hd (inouts0 n1)) (λn1. inouts0 n1!(Suc 0)) n =
      latch-rec-calc-output (λn1. hd (inouts0 n1)) (λn1. inouts0 n1!(Suc 0)) (Suc (n - Suc 0))
    using a5' by simp
    show ?thesis
      apply (simp only: 11)
      apply (simp)
      using 1 a3 by (simp add: a5')
  qed
next
  show ((n = 0 ∨ latch-rec-calc-output (λn1. hd (inouts0 n1)) (λn1. inouts0 n1!(Suc 0)) (n - Suc
0) = 0) ∧ hd (inouts0 n) = 0 →
  latch-rec-calc-output (λn1. hd (inouts0 n1)) (λn1. inouts0 n1!(Suc 0)) n = 0) ∧
  (¬ inouts0 n!(Suc 0) = 0 → latch-rec-calc-output (λn1. hd (inouts0 n1)) (λn1. inouts0 n1!(Suc
0)) n = 0))
  proof (cases)
    assume a4: n = 0
    then show ?thesis
      by simp
  next
    assume a4: ¬ n = 0
    then have a4': n > 0
      by simp
    show ?thesis
      apply (rule conjI, clarify)
      apply (metis Suc-pred a4 a4' latch-rec-calc-output.simps(2))
      using a4 a4' less-imp-Suc-add by fastforce
  qed
qed

```

abbreviation $\text{latch-simp-pat-f} \equiv (\lambda x na. [\text{if } (0 < na \wedge \neg \text{latch-rec-calc-output } (\lambda n1. \text{hd } (x n1)) (\lambda n1. x n1!(\text{Suc } 0)) (na - \text{Suc } 0) = 0 \vee \neg \text{hd } (x na) = 0) \wedge x na!(\text{Suc } 0) = 0 \text{ then } 1 \text{ else } 0])$

abbreviation $\text{latch-simp-pat-f}' \equiv (\lambda x na. [\text{latch-rec-calc-output } (\lambda n1. \text{hd } (x n1)) (\lambda n1. x n1!(\text{Suc } 0)) (na)])$

lemma $\text{latch-simp-pat-f-eq}$:

```

latch-simp-pat-f = latch-simp-pat-f'
proof -
  have 1: ∀ x na. latch-simp-pat-f x na = latch-simp-pat-f' x na
    apply (rule allI)+
    apply (induct-tac na)
  proof -
    fix x na
    have 1: [(if (0 < 0 ∧ ¬ latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (0 - Suc 0))
= 0 ∨
  ¬ hd (x 0) = 0) ∧
  x 0!(Suc 0) = 0

```



```

    then 1 else 0] = [(if ¬ hd (x 0) = 0 ∧ x 0!(Suc 0) = 0 then 1 else 0)]
  by (simp)
  have 2: [latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) 0] =
    [(if ¬ hd (x 0) = 0 ∧ x 0!(Suc 0) = 0 then 1 else 0)]
  by (simp)
  show [if (0 < 0 ∧ ¬ latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (0 - Suc 0) =
0 ∨
    ¬ hd (x 0) = 0) ∧
    x 0!(Suc 0) = 0
    then 1 else 0] =
    [latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) 0]
  using 1 2 by (simp)
next
fix x na n
assume a1: [if (0 < n ∧
  ¬ latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (n - Suc 0) = 0 ∨
  ¬ hd (x n) = 0) ∧ x n!(Suc 0) = 0
  then 1 else 0] =
  [latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) n]
show [if (0 < Suc n ∧ ¬ latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (Suc n -
Suc 0) = 0 ∨
  ¬ hd (x (Suc n)) = 0) ∧
  x (Suc n)!(Suc 0) = 0
  then 1 else 0] =
  [latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (Suc n)]
using a1 latch-rec-calc-output-0-1 by force
qed
show ?thesis
using 1 by simp
qed

```

abbreviation *latch-simp-pat* \equiv *FBlock* (λx n. True) 2 1 *latch-simp-pat-f*

lemma *SimBlock-latch-simp*:

```

  SimBlock 2 1 latch-simp-pat
  apply (rule SimBlock-FBlock)
  apply (rule-tac x = λna. [0, 1] in exI)
  apply (rule-tac x = λna. [0] in exI)
  apply (simp)
  by simp

```

abbreviation *latch-simp-pat'* \equiv *FBlock* (λx n. True) 2 1 *latch-simp-pat-f'*

lemma *SimBlock-latch-simp'*:

```

  SimBlock 2 1 latch-simp-pat'
  using SimBlock-latch-simp latch-simp-pat-f-eq
  by simp

```

lemma *latch-simp*:

```

  latch = latch-simp-pat'
  proof -

```

```

  have f1: (UnitDelay 0 (*3*) ||B Id) = (FBlock (λx n. True) (2) (2)
    (λx n. [if n = 0 then 0 else hd(x (n-1)), hd(tl(x n))]))
  using UnitDelay-Id-parallel-comp by (simp)

```

```

have simblock-f1: SimBlock 2 2 (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ))
by (metis (no-types, lifting) SimBlock-Id SimBlock-FBlock-parallel-comp SimBlock-UnitDelay
  Suc-1 Suc-eq-plus1 UnitDelay-Id-parallel-comp UnitDelay-def Id-def)

have f2: ((UnitDelay 0 (*3*)  $\parallel_B$  Id) ;; (LopOR 2 (*1*))) = (FBlock ( $\lambda x n. \text{True}$ ) (2) (2)
  ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ )) ;; (LopOR 2 (*1*))
by (simp add: UnitDelay-Id-parallel-comp)
have f2-0: ... = FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  ( $f\text{-LopOR}$  o ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ))
using LopOR-def FBlock-seq-comp SimBlock-LopOR simblock-f1 by auto
have f2-1: ... = FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  ( $\lambda x n. [\text{if } (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0]$ )
proof -
  have  $\forall x n. ((f\text{-LopOR}$  o ( $\lambda x n. [\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), \text{hd}(\text{tl}(x n))]$ ))  $x n$ 
    = ( $\lambda x n. [\text{if } (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0]$ )  $x n$ )
    using f-LopOR-def by auto
  then show ?thesis
    by presburger
qed
have simblock-f2: SimBlock 2 1 (FBlock ( $\lambda x n. \text{True}$ ) (2) (1)
  ( $\lambda x n. [\text{if } (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0]$ ))
by (metis (no-types, lifting) LopOR-def SimBlock-LopOR SimBlock-FBlock-seq-comp f2-0 f2-1
  pos2 simblock-f1)

have f3: (Id ;; LopNOT (*2*)) = (FBlock ( $\lambda x n. \text{True}$ ) (1) (1) ( $f\text{-LopNOT}$  o  $f\text{-Id}$ ))
by (metis LopNOT-def One-nat-def FBlock-seq-comp SimBlock-Id SimBlock-LopNOT
  simu-contract-real.Id-def)
then have f3-0: ... = (FBlock ( $\lambda x n. \text{True}$ ) (1) (1)
  ( $\lambda x n. [\text{if } \text{hd}(x n) = 0 \text{ then } 1 \text{ else } 0]$ ))
proof -
  have  $\forall x n. ((f\text{-LopNOT}$  o  $f\text{-Id}$ )  $x n$  = ( $\lambda x n. [\text{if } \text{hd}(x n) = 0 \text{ then } 1 \text{ else } 0]$ )  $x n$ )
    by (simp add: f-Id-def f-LopNOT-def)
  then show ?thesis
    by presburger
qed
have simblock-f3: SimBlock 1 1 (FBlock ( $\lambda x n. \text{True}$ ) (1) (1)
  ( $\lambda x n. [\text{if } \text{hd}(x n) = 0 \text{ then } 1 \text{ else } 0]$ ))
by (metis LopNOT-def SimBlock-Id SimBlock-LopNOT SimBlock-FBlock-seq-comp f3 f3-0 Id-def)

let ?P = ( $\lambda x n. [\text{if } (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0]$ )
let ?Q = ( $\lambda x n. [\text{if } \text{hd}(x n) = 0 \text{ then } 1 \text{ else } 0]$ )
have f4: (((UnitDelay 0 (*3*)  $\parallel_B$  Id) ;; (LopOR 2 (*1*)))  $\parallel_B$  (Id ;; LopNOT (*2*)))
  = (FBlock ( $\lambda x n. \text{True}$ ) (2) (1) ?P)  $\parallel_B$  (FBlock ( $\lambda x n. \text{True}$ ) (1) (1) ?Q)
using f2 f2-0 f2-1 f3 f3-0 by auto
then have f4-0: ... = FBlock ( $\lambda x n. \text{True}$ ) (2+1) (1+1)
  ( $\lambda x n. (((?P$  o ( $\lambda x n n. \text{take } 2 (x n)$ )))  $x n$ )
    • (( $?Q$  o ( $\lambda x n n. \text{drop } 2 (x n)$ )))  $x n$ )
using SimBlock-UnitDelay SimBlock-Id SimBlock-LopOR SimBlock-LopNOT simblock-f1 simblock-f2
simblock-f3
by (simp add: FBlock-parallel-comp f-sim-blocks)
then have f4-1: ... = FBlock ( $\lambda x n. \text{True}$ ) 3 2
  ( $\lambda x n. (((?P$  o ( $\lambda x n n. \text{take } 2 (x n)$ )))  $x n$ )
    • (( $?Q$  o ( $\lambda x n n. \text{drop } 2 (x n)$ )))  $x n$ )
using Suc-eq-plus1 nat-1-add-1 numeral-2-eq-2 numeral-3-eq-3 by presburger

```

```

have f4-2: FBlock (λx n. True) 3 2
  (λx n. (((?P ∘ (λxx nn. take 2 (xx nn)))) x n)
    • ((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n))
= FBlock (λx n. True) 3 2
  (λx n. ([if (n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0 then 1::real else 0,
    if (x n)!2 = 0 then 1 else 0]))
proof -
  have 1: ∀ (x::nat ⇒ real list) n::nat. length(x n) > 2 →
    (((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n
    = (λx n. [if (x n)!2 = 0 then 1 else 0]) x n)
    apply (auto)
    apply (simp add: hd-drop-conv-nth)
    by (simp add: hd-drop-conv-nth)
  have 2: ∀ (x::nat ⇒ real list) n::nat. ((λx n. (((?P ∘ (λxx nn. take 2 (xx nn)))) x n)
    • ((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n)) x n
    = (λx n. (((λx n. [if (n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0 then 1::real else 0]) x n)
    • ((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n)) x n)
    apply (auto)
    apply (metis append-take-drop-id hd-append2 take-eq-Nil zero-not-eq-two)
    apply (metis Suc-1 append-take-drop-id hd-append2 take-eq-Nil take-tl zero-neq-one)
    apply (metis Suc-1 append-take-drop-id hd-append2 take-eq-Nil take-tl zero-neq-one)
    apply (metis Suc-1 hd-conv-nth less-numeral-extra(1) nth-take take-eq-Nil take-tl zero-neq-one)
    apply (metis Suc-1 append-take-drop-id hd-append2 take-eq-Nil take-tl zero-neq-one)
    apply (metis append-take-drop-id hd-append2 take-eq-Nil zero-not-eq-two)
    by (metis Suc-1 append-take-drop-id hd-append2 take-eq-Nil take-tl zero-neq-one)
  have 3: ∀ (x::nat ⇒ real list) n::nat. length(x n) > 2 →
    ((λx n. (((λx n. [if (n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0 then 1::real else 0]) x n)
    • ((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n)) x n
    = (λx n. ([if (n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0 then 1::real else 0,
    if (x n)!2 = 0 then 1 else 0]) x n)
    using hd-drop-m by simp
  have 4: ∀ (x::nat ⇒ real list) n::nat. length(x n) > 2 →
    ((λx n. (((?P ∘ (λxx nn. take 2 (xx nn)))) x n)
    • ((?Q ∘ (λxx nn. drop 2 (xx nn)))) x n)) x n
    = (λx n. ([if (n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0 then 1::real else 0,
    if (x n)!2 = 0 then 1 else 0]) x n)
    using 1 2 by simp
  show ?thesis
    apply (simp add: FBlock-def)
    apply (rel-simp)
    apply (rule iffI)
    apply (clarify)
    defer
    apply (clarify)
    defer
  proof -
    fix okv inoutsv::nat ⇒ real list and okv' inoutsv'::nat ⇒ real list and x::nat
    assume a1: ∀ x. (hd (drop 2 (inoutsv x)) = 0 →
      (0 < x ∧ ¬ hd (take 2 (inoutsv (x - Suc 0)))) = 0 → length(inoutsv x) = 3 ∧ length(inoutsv'
x) = 2 ∧ [1, 1] = inoutsv' x) ∧
      (¬ hd (tl (take 2 (inoutsv x))) = 0 → length(inoutsv x) = 3 ∧ length(inoutsv' x) = 2 ∧
[1, 1] = inoutsv' x) ∧
      ((x = 0 ∨ hd (take 2 (inoutsv (x - Suc 0))) = 0) ∧ hd (tl (take 2 (inoutsv x))) = 0 →
length(inoutsv x) = 3 ∧ length(inoutsv' x) = 2 ∧ [0, 1] = inoutsv' x) ∧
      (¬ hd (drop 2 (inoutsv x)) = 0 →

```

$(0 < x \wedge \neg \text{hd}(\text{take } 2(\text{inouts}_v(x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd}(\text{tl}(\text{take } 2(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $((x = 0 \vee \text{hd}(\text{take } 2(\text{inouts}_v(x - \text{Suc } 0))) = 0) \wedge \text{hd}(\text{tl}(\text{take } 2(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))$
from *a1* **have** *len-3*: $\forall na. \text{length}(\text{inouts}_v na) = 3$
by (*meson neq0-conv*)
from *len-3* **have** *hd-drop*: $(\text{hd}(\text{drop } 2(\text{inouts}_v x)) = \text{inouts}_v x!(2))$
by (*simp add: hd-drop-conv-nth*)
have *hd-take*: $\text{hd}(\text{take } 2(\text{inouts}_v(x - \text{Suc } 0))) = \text{hd}(\text{inouts}_v(x - \text{Suc } 0))$
by (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral*)
have *hd-tl-take*: $\text{hd}(\text{tl}(\text{take } 2(\text{inouts}_v x))) = \text{hd}(\text{tl}(\text{inouts}_v x))$
by (*metis Suc-1 hd-conv-nth less-numeral-extra(1) nth-take take-eq-Nil take-tl zero-neq-one*)
show $(\text{inouts}_v x!(2) = 0 \longrightarrow$
 $(0 < x \wedge \neg \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd}(\text{tl}(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $((x = 0 \vee \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0) \wedge \text{hd}(\text{tl}(\text{inouts}_v x)) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 1] = \text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v x!(2) = 0 \longrightarrow$
 $(0 < x \wedge \neg \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd}(\text{tl}(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $((x = 0 \vee \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0) \wedge \text{hd}(\text{tl}(\text{inouts}_v x)) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))$
using *a1* *hd-drop* *hd-take* *hd-tl-take* **by** *presburger*
next
fix *ok_v::bool* **and** *inouts_v::nat* \Rightarrow *real list* **and** *ok_v'::bool* **and** *inouts_v'::nat* \Rightarrow *real list* **and** *x::nat*
assume *a1*: $(\forall x. (\text{inouts}_v x!(2) = 0 \longrightarrow$
 $(0 < x \wedge \neg \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd}(\text{tl}(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $((x = 0 \vee \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0) \wedge \text{hd}(\text{tl}(\text{inouts}_v x)) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 1] = \text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v x!(2) = 0 \longrightarrow$
 $(0 < x \wedge \neg \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd}(\text{tl}(\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $((x = 0 \vee \text{hd}(\text{inouts}_v(x - \text{Suc } 0))) = 0) \wedge \text{hd}(\text{tl}(\text{inouts}_v x)) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))$
from *a1* **have** *len-3*: $\forall na. \text{length}(\text{inouts}_v na) = 3$
by (*meson neq0-conv*)
from *len-3* **have** *hd-drop*: $(\text{hd}(\text{drop } 2(\text{inouts}_v x)) = \text{inouts}_v x!(2))$
by (*simp add: hd-drop-conv-nth*)
have *hd-take*: $\text{hd}(\text{take } 2(\text{inouts}_v(x - \text{Suc } 0))) = \text{hd}(\text{inouts}_v(x - \text{Suc } 0))$
by (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral*)
have *hd-tl-take*: $\text{hd}(\text{tl}(\text{take } 2(\text{inouts}_v x))) = \text{hd}(\text{tl}(\text{inouts}_v x))$
by (*metis Suc-1 hd-conv-nth less-numeral-extra(1) nth-take take-eq-Nil take-tl zero-neq-one*)
show $((\text{hd}(\text{drop } 2(\text{inouts}_v x)) = 0 \longrightarrow$

$$\begin{aligned}
& (0 < x \wedge \neg \text{hd} (\text{take } 2 (\text{inouts}_v (x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\neg \text{hd} (\text{tl} (\text{take } 2 (\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = \\
& 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& ((x = 0 \vee \text{hd} (\text{take } 2 (\text{inouts}_v (x - \text{Suc } 0))) = 0) \wedge \text{hd} (\text{tl} (\text{take } 2 (\text{inouts}_v x))) = 0 \\
\longrightarrow & \\
& \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 1] = \text{inouts}_v' x) \wedge \\
& (\neg \text{hd} (\text{drop } 2 (\text{inouts}_v x)) = 0 \longrightarrow \\
& (0 < x \wedge \neg \text{hd} (\text{take } 2 (\text{inouts}_v (x - \text{Suc } 0))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{hd} (\text{tl} (\text{take } 2 (\text{inouts}_v x))) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = \\
& 2 \wedge [1, 0] = \text{inouts}_v' x) \wedge \\
& ((x = 0 \vee \text{hd} (\text{take } 2 (\text{inouts}_v (x - \text{Suc } 0))) = 0) \wedge \text{hd} (\text{tl} (\text{take } 2 (\text{inouts}_v x))) = 0 \\
\longrightarrow & \\
& \text{length}(\text{inouts}_v x) = 3 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \\
& \text{by (simp add: a1 hd-drop hd-take hd-tl-take)} \\
& \text{qed} \\
& \text{qed} \\
& \text{then have f4-3: } (((\text{UnitDelay } 0 (*3*) \parallel_B \text{Id}) ;; (\text{LopOR } 2 (*1*))) \parallel_B (\text{Id} ;; \text{LopNOT } (*2*))) \\
& = \text{FBlock } (\lambda x n. \text{True}) 3 2 \\
& (\lambda x n. ([if (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0, \\
& \text{if } (x n)!2 = 0 \text{ then } 1 \text{ else } 0])) \\
& \text{using f4 f4-0 f4-1 by simp} \\
& \text{have simblock-f4: SimBlock } 3 2 (\text{FBlock } (\lambda x n. \text{True}) 3 2 \\
& (\lambda x n. ([if (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0, \\
& \text{if } (x n)!2 = 0 \text{ then } 1 \text{ else } 0]))) \\
& \text{by (metis (no-types, lifting) One-nat-def SimBlock-FBlock-parallel-comp Suc-1 Suc-eq-plus1 f4} \\
& \text{f4-3 numeral-3-eq-3 simblock-f2 simblock-f3)} \\
& \text{have f5: } (((\text{UnitDelay } 0 (*3*) \parallel_B \text{Id}) ;; (\text{LopOR } 2 (*1*))) \\
& \parallel_B \\
& (\text{Id} ;; \text{LopNOT } (*2*))) \\
&) ;; (\text{LopAND } 2) (*Latch-1*)) = \\
& \text{FBlock } (\lambda x n. \text{True}) 3 2 \\
& (\lambda x n. ([if (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0, \\
& \text{if } (x n)!2 = 0 \text{ then } 1 \text{ else } 0])) ;; (\text{LopAND } 2) \\
& \text{using f4-3 by simp} \\
& \text{then have f5-0: } \dots = \text{FBlock } (\lambda x n. \text{True}) 3 1 \\
& (\text{f-LopAND } o (\lambda x n. ([if (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real else } 0, \\
& \text{if } (x n)!2 = 0 \text{ then } 1 \text{ else } 0]))) \\
& \text{by (metis (no-types, lifting) LopAND-def One-nat-def FBlock-seq-comp SimBlock-LopAND} \\
& \text{SimBlock-FBlock-parallel-comp Suc-1 Suc-eq-plus1 f4 f4-3 numeral-3-eq-3 pos2 simblock-f2} \\
& \text{simblock-f3)} \\
& \text{then have f5-1: } \dots = \text{FBlock } (\lambda x n. \text{True}) 3 1 \\
& (\lambda x n. ([if ((n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0) \wedge (x n)!2 = 0 \text{ then } 1::\text{real else } 0])) \\
& \text{proof -} \\
& \text{have } \forall x n. (\text{f-LopAND } o (\lambda x n. ([if (n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0 \text{ then } 1::\text{real} \\
& \text{else } 0, \\
& \text{if } (x n)!2 = 0 \text{ then } 1 \text{ else } 0]))) x n \\
& = (\lambda x n. ([if ((n > 0 \wedge \text{hd}(x (n-1)) \neq 0) \vee \text{hd}(\text{tl}(x n)) \neq 0) \wedge (x n)!2 = 0 \text{ then } 1::\text{real else} \\
& 0])) x n \\
& \text{by (simp add: f-LopAND-def)} \\
& \text{then show ?thesis} \\
& \text{apply (simp add: FBlock-def)} \\
& \text{apply (rel-simp)}
\end{aligned}$$

```

apply (simp add: f-LopAND-def)
apply (rule iffI)
apply (clarify)
using neg0-conv apply blast
apply (clarify)
by blast
qed
have simblock-f5: SimBlock 3 1 (FBlock (λx n. True) 3 1
  (λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])]))
using simblock-f4
by (metis (no-types, lifting) LopAND-def SimBlock-LopAND SimBlock-FBlock-seq-comp f5-0 f5-1
pos2)

have f6: (((UnitDelay 0 (*3*) ||B Id) ;; (LopOR 2 (*1*)))
  ||B
  (Id ;; LopNOT (*2*))) ;; (LopAND 2) (*Latch-1*) ;; Split2)
  = (FBlock (λx n. True) 3 1
  (λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0]))
  ;; Split2)
using f5 f5-0 f5-1 by (simp add: RA1)
then have f6-0: ... = (FBlock (λx n. True) 3 2 (f-Split2 o
  (λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])]))
using Split2-def FBlock-seq-comp simblock-f5 by (metis (no-types, lifting) SimBlock-Split2)
then have f6-1: ... = (FBlock (λx n. True) 3 2
  ((λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0,
  if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])))
proof –
have  $\forall n f. [if (0 < n \wedge \neg hd (f (n - 1)) = (0::real) \vee \neg hd (tl (f n)) = 0) \wedge$ 
   $f n!(2) = (0::real) \text{ then } 1 \text{ else } 0, \text{ if } (0 < n \wedge \neg hd (f (n - 1)) = 0 \vee$ 
   $\neg hd (tl (f n)) = 0) \wedge f n!(2) = (0::real) \text{ then } 1 \text{ else } 0] =$ 
  (f-Split2  $\circ$  ( $\lambda f n. [if (0 < n \wedge \neg hd (f (n - 1)) = 0 \vee \neg hd (tl (f n)) = 0) \wedge$ 
   $f n!(2) = (0::real) \text{ then } 1 \text{ else } 0]$ )) f n
by (simp add: f-Split2-def)
then show ?thesis
by presburger
qed
have simblock-f6: SimBlock 3 2 (FBlock (λx n. True) 3 2
  ((λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0,
  if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])))
using simblock-f5 SimBlock-Split2
by (smt SimBlock-FBlock-seq-comp Split2-def f6-0 f6-1)
let ?f6 = (FBlock (λx n. True) 3 2
  ((λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0,
  if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])))
have inps-f6: inps ?f6 = 3
using inps-P simblock-f6 by blast
have outps-f6: outps ?f6 = 2
using outps-P simblock-f6 by blast

have f7: latch = ?f6 fD (0,0)
using f6 f6-0 f6-1 latch-def by simp
have is-solution-f7: is-Solution 0 0 3 2
  ((λx n. ([if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0,
  if ((n > 0 ∧ hd(x (n-1)) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])))
  ( $\lambda \text{inouts}_0::\text{nat} \Rightarrow \text{real list}$ ).  $\lambda na. \text{latch-rec-calc-output}$ 

```

```

      (λn1. hd(inouts0 n1)) (λn1. (inouts0 n1)!1) na)
apply (simp add: is-Solution-def)
apply (rule allI)
apply (clarify)
apply (simp add: f-PreFD-def)
using latch-rec-calc-output-is-a-solution by blast
have unique-f7: Solvable-unique 0 0 3 2
  (λx n. ([if ((n > 0 ∧ hd(x (n-1))) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0,
    if ((n > 0 ∧ hd(x (n-1))) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0]))
apply (simp add: Solvable-unique-def)
apply (rule allI, clarify, simp add: f-PreFD-def)
apply (rule ex-ex1I)
apply (rule-tac x = λna. latch-rec-calc-output (λn1. hd(inouts0 n1)) (λn1. (inouts0 n1)!1) na in
exI)
apply (simp)
apply (rule allI)
using latch-rec-calc-output-is-a-solution apply blast
proof –
  fix inouts0::nat ⇒ real list and xx y ::nat ⇒ real
  assume a1: ∀ n. ((0 < n ∧ ¬ xx (n - Suc 0) = 0 ∨ ¬ hd (inouts0 n) = 0) ∧
    inouts0 n!(Suc 0) = 0 → xx n = 1) ∧
    ((n = 0 ∨ xx (n - Suc 0) = 0) ∧ hd (inouts0 n) = 0 → xx n = 0) ∧
    (¬ inouts0 n!(Suc 0) = 0 → xx n = 0)
  assume a2: ∀ n. ((0 < n ∧ ¬ y (n - Suc 0) = 0 ∨ ¬ hd (inouts0 n) = 0) ∧
    inouts0 n!(Suc 0) = 0 → y n = 1) ∧
    ((n = 0 ∨ y (n - Suc 0) = 0) ∧ hd (inouts0 n) = 0 → y n = 0) ∧
    (¬ inouts0 n!(Suc 0) = 0 → y n = 0)
  have 1: ∀ n. xx n = y n
  apply (rule allI)
  proof –
    fix n::nat
    show xx n = y n
    proof (induct n)
      case 0
      then show ?case
      using a1 a2 by metis
    next
      case (Suc n) note IH = this
      then show ?case
      using a1 a2 by (metis One-nat-def diff-Suc-1 zero-less-Suc)
    qed
  qed
  show xx = y
  using 1 fun-eq by (blast)
qed
have f7-0:
  ?f6 fD (0,0) = (FBlock (λx n. True) (3-1) (2-1)
    (λx na. ((f-PostFD 0)
      o (λx n. ([if ((n > 0 ∧ hd(x (n-1))) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else
0,
        if ((n > 0 ∧ hd(x (n-1))) ≠ 0) ∨ hd(tl(x n)) ≠ 0) ∧ (x n)!2 = 0 then 1::real else 0])))
    o (f-PreFD ((λ(inouts0::nat ⇒ real list). λna. latch-rec-calc-output
      (λn1. hd(inouts0 n1)) (λn1. (inouts0 n1)!1) na) x) 0)) x na)
  using FBlock-feedback' f7 is-solution-f7 unique-f7 simblock-f6 by blast
then have f7-1: ... = FBlock (λx n. True) 2 1

```

```

(λx na. [if (0 < na ∧
  ¬ latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) (na - Suc 0) = 0
  ∨ ¬ hd (x na) = 0) ∧ x na!(Suc 0) = 0
  then 1 else 0])
by (simp (no-asm) add: f-PreFD-def f-PostFD-def)
show ?thesis
using f7 f7-0 f7-1 latch-simp-pat-f-eq by (simp)
qed

```

C.3.1 Verification

latch-req-00: if R is true, then the output is always false.

lemma *latch-req-00*:

```

((∀ n::nat · (
  «(λx n. ((hd(x n) = 0 ∨ hd(x n) = 1) ∧ (hd(tl(x n)) = 0 ∨ hd(tl(x n)) = 1)))»
  (&inouts)a («n»)a::sim-state upred)

```

⊢_n

```

((∀ n::nat ·
  ((#u($inouts («n»)a)) =u «2») ∧
  ((#u($inouts' («n»)a)) =u «1») ∧
  (headu(tailu($inouts («n»)a)) ≠u 0) ⇒ (headu($inouts' («n»)a) =u 0))
)) ⊆ latch

```

using *latch-simp* **apply** (simp add: latch-def)

proof –

```

show (∀ n · «λx n. (hd (x n) = 0 ∨ hd (x n) = 1) ∧ (hd (tl (x n)) = 0 ∨ hd (tl (x n)) = 1)»
  (&inouts)a(«n»)a ⊢n
  (∀ n · #u($inouts («n»)a) =u «2» ∧
    #u($inouts' («n»)a) =u «Suc 0» ∧ headu(tailu($inouts («n»)a)) ≠u 0 ⇒
    headu($inouts' («n»)a) =u 0)

```

⊆

```

FBlock (λx n. True) 2 (Suc 0)
(λx na. [latch-rec-calc-output (λn1. hd (x n1)) (λn1. x n1!(Suc 0)) na])

```

apply (simp add: FBlock-def)

apply (rule ndesign-refine-intro)

apply simp

apply (rel-simp)

proof –

fix *inouts_v* *inouts_v'*::nat ⇒ real list **and** *x*::nat

assume *a1*: ∀ *x*. (hd (inouts_v *x*) = 0 ∨ hd (inouts_v *x*) = 1) ∧ (hd (tl (inouts_v *x*)) = 0 ∨
hd (tl (inouts_v *x*)) = 1)

assume *a2*: ∀ *x*. length(inouts_v *x*) = 2 ∧
length(inouts_v' *x*) = Suc 0 ∧

[latch-rec-calc-output (λn1. hd (inouts_v n1)) (λn1. inouts_v n1!(Suc 0)) *x*] = inouts_v' *x*

assume *a3*: ¬ hd (tl (inouts_v *x*)) = 0

have *1*: ¬ inouts_v *x*!(Suc 0) = 0

using *a2 a3*

by (metis One-nat-def Suc-1 diff-Suc-1 diff-is-0-eq hd-conv-nth length-tl
less-numeral-extra(1) list.size(3) not-one-le-zero nth-tl)

have *2*: inouts_v' *x* = [0]

using *a2 1*

by (metis (mono-tags, lifting) latch-rec-calc-output.elims)

then show hd (inouts_v' *x*) = 0

by (simp)

qed

qed

C.4 System: *post-landing-finalize*

post-mode is a part of block compositions from the input *mode* to the three-way AND logic block.

definition *post-mode* \equiv
 $(\text{Split2 } (* \text{ mode is split into two } *) \ ; \ ;$
 $($
 $((\text{UnitDelay } 0 \ (*IC = 0, r=1/10s*) \ \|_B \ \text{Const } 4 \ (*\text{landing, uint32}(4), r=1/10s*)) \ ; \ ; \ \text{RopEQ})$
 $\|_B$
 $((\text{Id} \ \|_B \ \text{Const } 8 \ (*\text{ground, uint32}(8), r=1/10s*)) \ ; \ ; \ \text{RopEQ})$
 $)$
 $)$

lemma *post-mode-simp*:

post-mode = $(\text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (2)$
 $(\lambda x \ n. \ ([\text{if } (n > 0 \wedge \text{hd}(x \ (n-1)) = 4) \ \text{then } 1::\text{real} \ \text{else } 0, \ \text{if } \text{hd}(x \ n) = 8 \ \text{then } 1 \ \text{else } 0])))$
proof –

have *f1*: $(\text{UnitDelay } 0 \ (*IC = 0, r=1/10s*) \ \|_B \ \text{Const } 4 \ (*\text{landing, uint32}(4), r=1/10s*))$
 $= \text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (2)$
 $(\lambda x \ n. \ (((f\text{-UnitDelay } 0 \ \circ \ (\lambda xx \ nn. \ \text{take } 1 \ (xx \ nn)))) \ x \ n) \ \bullet$
 $((f\text{-Const } 4 \ \circ \ (\lambda xx \ nn. \ \text{drop } 1 \ (xx \ nn)))) \ x \ n))$
using *SimBlock-UnitDelay SimBlock-Const* **apply** (*simp add: FBlock-parallel-comp f-sim-blocks*)
by (*simp add: numeral-2-eq-2*)
have *f1-0*: $\dots = \text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (2)$
 $(\lambda x \ n. \ ([\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd}(x \ (n-1)), 4]))$
using *f-UnitDelay-def f-Const-def* **apply** (*auto*)
proof –
 $\{ \text{fix } nn :: \text{nat} \ \text{and} \ rrs :: \text{nat} \Rightarrow \text{real list}$
 $\text{have } \forall rs \ n. \ \text{hd} \ (\text{take } n \ rs) = (\text{hd } rs::\text{real}) \vee \ \text{take } n \ rs = []$
 $\text{by} \ (\text{metis } \text{append-take-drop-id } \text{hd-append2})$
 $\text{then have } \text{FBlock } (\lambda f \ n. \ \text{True}) \ (\text{Suc } 0) \ 2 \ (\lambda f \ n. \ [\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (\text{take} \ (\text{Suc } 0) \ (f \ (n$
 $- 1))), 4])$
 $= \text{FBlock } (\lambda f \ n. \ \text{True}) \ (\text{Suc } 0) \ 2 \ (\lambda f \ n. \ [\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (f \ (n - 1)), 4]) \vee$
 $[\text{if } nn = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (\text{take} \ (\text{Suc } 0) \ (rrs \ (nn - 1))), 4] = [\text{if } nn = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (rrs$
 $(nn - 1)), 4])$
 $\text{by force } \}$
 $\text{then show } \text{FBlock } (\lambda f \ n. \ \text{True}) \ (\text{Suc } 0) \ 2 \ (\lambda f \ n. \ [\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (\text{take} \ (\text{Suc } 0) \ (f \ (n -$
 $1))), 4])$
 $= \text{FBlock } (\lambda f \ n. \ \text{True}) \ (\text{Suc } 0) \ 2 \ (\lambda f \ n. \ [\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd} \ (f \ (n - 1)), 4])$
 by presburger
qed
have *simblock-f1*: $\text{SimBlock } 1 \ 2 \ (\text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (2)$
 $(\lambda x \ n. \ ([\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd}(x \ (n-1)), 4])))$
using *SimBlock-UnitDelay SimBlock-Const f1 f1-0* **apply** (*simp add: SimBlock-FBlock-parallel-comp f-sim-blocks*)
by (*smt One-nat-def SimBlock-FBlock-parallel-comp Suc-1 Suc-eq-plus1 add.right-neutral*)

have *f2*: $((\text{UnitDelay } 0 \ (*IC = 0, r=1/10s*) \ \|_B \ \text{Const } 4 \ (*\text{landing, uint32}(4), r=1/10s*)) \ ; \ ;$
 $\text{RopEQ}) =$
 $(\text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (2) \ (\lambda x \ n. \ ([\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd}(x \ (n-1)), 4]))) \ ; \ ; \ \text{RopEQ}$
using *f1 f1-0* **by** *simp*
then have *f2-0*: $\dots =$
 $(\text{FBlock } (\lambda x \ n. \ \text{True}) \ (1) \ (1) \ (f\text{-RopEQ} \ \circ \ (\lambda x \ n. \ ([\text{if } n = 0 \ \text{then } 0 \ \text{else } \text{hd}(x \ (n-1)), 4])))$

```

using simblock-f1 SimBlock-RopEQ FBlock-seq-comp by (simp add: RopEQ-def)
then have f2-1: ... = (FBlock ( $\lambda x n. \text{True}$ ) (1) (1)
( $\lambda x n. ([\text{if } (n > 0 \wedge \text{hd}(x (n-1)) = 4] \text{ then } 1::\text{real else } 0]$ )))
proof -
  have  $\forall x n. (f\text{-RopEQ } o (\lambda x n. ([\text{if } n = 0 \text{ then } 0 \text{ else } \text{hd}(x (n-1)), 4]))) x n$ 
    = ( $\lambda x n. ([\text{if } (n > 0 \wedge \text{hd}(x (n-1)) = 4] \text{ then } 1::\text{real else } 0]$ ) x n
    using f-RopEQ-def by auto
  then show ?thesis
    by presburger
qed
have simblock-f2: SimBlock 1 1 (FBlock ( $\lambda x n. \text{True}$ ) (1) (1)
( $\lambda x n. ([\text{if } (n > 0 \wedge \text{hd}(x (n-1)) = 4] \text{ then } 1::\text{real else } 0]$ )))
using f2 f2-0 f2-1 by (smt RopEQ-def SimBlock-FBlock-seq-comp SimBlock-RopEQ simblock-f1)

have f3: (Id ||B Const 8 (*ground, uint32(8), r=1/10s*))
= FBlock ( $\lambda x n. \text{True}$ ) (1) (2)
( $\lambda x n. (((f\text{-Id } o (\lambda xx nn. \text{take } 1 (xx nn))) x n) \bullet$ 
( $f\text{-Const } 8 \circ (\lambda xx nn. \text{drop } 1 (xx nn)))) x n$ )
using SimBlock-Id SimBlock-Const apply (simp add: FBlock-parallel-comp f-sim-blocks)
by (simp add: numeral-2-eq-2)
then have f3-0: ... = FBlock ( $\lambda x n. \text{True}$ ) (1) (2) ( $\lambda x n. ([\text{hd}(x n), 8]$ ))
proof -
  have  $\forall x n. ((\lambda x n. (((f\text{-Id } o (\lambda xx nn. \text{take } 1 (xx nn))) x n) \bullet$ 
( $f\text{-Const } 8 \circ (\lambda xx nn. \text{drop } 1 (xx nn)))) x n) x n$ 
    = ( $\lambda x n. ([\text{hd}(x n), 8]$ ) x n
    using f-Id-def f-Const-def
  proof -
    { fix rrs :: nat  $\Rightarrow$  real list and nn :: nat
      have  $\forall rs. \text{hd } (\text{take } 1 rs) = (\text{hd } rs::\text{real}) \vee rs = []$ 
        by (metis Suc-eq-plus1 add.left-neutral list.sel(1) take-Suc)
      then have ( $f\text{-Id } o (\lambda f n. \text{take } 1 (f n))) rrs nn \bullet (f\text{-Const } 8 \circ (\lambda f n. \text{drop } 1 (f n))) rrs nn =$ 
 $[\text{hd } (rrs nn), 8]$ 
        using f-Const-def f-Id-def by auto }
      then show ?thesis
        by fastforce
    qed
    then show ?thesis
      by simp
    qed
have simblock-f3: SimBlock 1 2 (FBlock ( $\lambda x n. \text{True}$ ) (1) (2) ( $\lambda x n. ([\text{hd}(x n), 8]$ ))
by (metis (no-types, lifting) One-nat-def SimBlock-Const SimBlock-Id SimBlock-FBlock-parallel-comp
Suc-1 Suc-eq-plus1 add commute f3 f3-0 simu-contract-real.Const-def simu-contract-real.Id-def)

have f4: ((Id ||B Const 8 (*ground, uint32(8), r=1/10s*)) ; ; RopEQ)
= FBlock ( $\lambda x n. \text{True}$ ) (1) (2) ( $\lambda x n. ([\text{hd}(x n), 8]$ )) ; ; RopEQ
using f3 f3-0 by simp
then have f4-0: ... = FBlock ( $\lambda x n. \text{True}$ ) (1) (1) ( $f\text{-RopEQ } o (\lambda x n. ([\text{hd}(x n), 8]$ ))
using simblock-f3 SimBlock-RopEQ FBlock-seq-comp by (simp add: RopEQ-def)
then have f4-1: ... = FBlock ( $\lambda x n. \text{True}$ ) (1) (1) ( $\lambda x n. ([\text{if } \text{hd}(x n) = 8 \text{ then } 1 \text{ else } 0]$ ))
using f-RopEQ-def by (metis (mono-tags, lifting) comp-apply list.sel(1) list.sel(3))
have simblock-f4: SimBlock 1 1
(FBlock ( $\lambda x n. \text{True}$ ) (1) (1) ( $\lambda x n. ([\text{if } \text{hd}(x n) = 8 \text{ then } 1 \text{ else } 0]$ ))
using simblock-f3 SimBlock-RopEQ by (metis RopEQ-def SimBlock-FBlock-seq-comp f4-0 f4-1)

```

```

have f5: (
  ((UnitDelay 0 (*IC = 0, r=1/10s*) ||B Const 4 (*landing, uint32(4), r=1/10s*)) ;; RopEQ)
  ||B
  ((Id ||B Const 8 (*ground, uint32(8), r=1/10s*)) ;; RopEQ))
= (FBlock (λx n. True) (1) (1) (λx n. ([if (n > 0 ∧ hd(x (n-1)) = 4) then 1::real else 0])))
  ||B
  (FBlock (λx n. True) (1) (1) (λx n. ([if hd(x n) = 8 then 1 else 0])))
using f2 f2-1 f4 f4-1 f2-0 f4-0 by auto
then have f5-0: ... = FBlock (λx n. True) (2) (2)
  (λx n. (((λx n. ([if (n > 0 ∧ hd(x (n-1)) = 4) then 1::real else 0]))
    ◦ (λxx nn. take 1 (xx nn))) x n) •
  ((λx n. ([if hd(x n) = 8 then 1 else 0]))
    ◦ (λxx nn. drop 1 (xx nn)))) x n)
using simblock-f2 simblock-f4 apply (simp add: FBlock-parallel-comp f-sim-blocks)
by (simp add: numeral-2-eq-2)
then have f5-1: ... = FBlock (λx n. True) (2) (2)
  (λx n. ([if (n > 0 ∧ hd(x (n-1)) = 4) then 1::real else 0, if (x n)! = 8 then 1 else 0]))
proof –
  show ?thesis
    apply (simp add: FBlock-def)
    apply (rel-simp)
    apply (rule conjI)
    apply (clarify)
    apply (rule conjI)
    apply (clarify)
    apply (rule iffI)
    apply (clarify)
    apply (subgoal-tac ∀ x. length(inoutsv x) = 2)
    apply (rule conjI)
    apply (clarify)
    using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
    using hd-drop-m hd-take-m apply simp
    using neq0-conv apply blast
    apply (clarify)
    apply (subgoal-tac ∀ x. length(inoutsv x) = 2)
    apply (rule conjI)
    apply (clarify)
    using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
    using hd-drop-m hd-take-m apply simp
    using neq0-conv apply blast
    apply (clarify)
    apply (rule iffI)
    apply (clarify)
    apply (subgoal-tac ∀ x. length(inoutsv x) = 2)
    apply (rule conjI)
    apply (clarify)
    using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
    using hd-drop-m hd-take-m apply simp

```

```

using neq0-conv apply blast
apply (clarify)
apply (rule conjI)
apply (clarify)
apply (rule iffI)
apply (clarify)
apply (subgoal-tac  $\forall x. \text{length}(\text{inouts}_v\ x) = 2$ )
apply (rule conjI)
apply (clarify)
using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
using hd-drop-m hd-take-m apply simp
using neq0-conv apply blast
apply (clarify)
apply (subgoal-tac  $\forall x. \text{length}(\text{inouts}_v\ x) = 2$ )
apply (rule conjI)
apply (clarify)
using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
using hd-drop-m hd-take-m apply simp
using neq0-conv apply blast
apply (clarify)
apply (rule iffI)
apply (clarify)
apply (subgoal-tac  $\forall x. \text{length}(\text{inouts}_v\ x) = 2$ )
apply (rule conjI)
apply (clarify)
using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
using hd-drop-m hd-take-m apply simp
apply metis
using neq0-conv apply blast
apply (clarify)
apply (subgoal-tac  $\forall x. \text{length}(\text{inouts}_v\ x) = 2$ )
apply (rule conjI)
apply (clarify)
using hd-drop-m hd-take-m apply (metis Suc-1 Suc-eq-plus1 add.left-neutral lessI)
using hd-drop-m hd-take-m apply simp
apply metis
using neq0-conv by blast
qed
have simblock-f5: SimBlock 2 2 (FBlock ( $\lambda x\ n. \text{True}$ ) (2) (2)
  ( $\lambda x\ n. (([\text{if } (n > 0 \wedge \text{hd}(x\ (n-1)) = 4] \text{ then } 1::\text{real} \text{ else } 0, \text{if } (x\ n)!1 = 8 \text{ then } 1 \text{ else } 0])$ )))
using simblock-f2 simblock-f4 SimBlock-FBlock-parallel-comp f5 f5-0 f5-1
by (metis (no-types, lifting) one-add-one)

have f6: post-mode = Split2 ;; (FBlock ( $\lambda x\ n. \text{True}$ ) (2) (2)
  ( $\lambda x\ n. (([\text{if } (n > 0 \wedge \text{hd}(x\ (n-1)) = 4] \text{ then } 1::\text{real} \text{ else } 0, \text{if } (x\ n)!1 = 8 \text{ then } 1 \text{ else } 0])$ )))
using f5 f5-0 f5-1 post-mode-def by auto
then have f6-0: ... = (FBlock ( $\lambda x\ n. \text{True}$ ) (1) (2) (
  ( $\lambda x\ n. (([\text{if } (n > 0 \wedge \text{hd}(x\ (n-1)) = 4] \text{ then } 1::\text{real} \text{ else } 0, \text{if } (x\ n)!1 = 8 \text{ then } 1 \text{ else } 0])$ ))) o
f-Split2)
using SimBlock-Split2 simblock-f5 by (simp add: FBlock-seq-comp f-sim-blocks)
then have f6-1: ... = (FBlock ( $\lambda x\ n. \text{True}$ ) (1) (2)
  ( $\lambda x\ n. (([\text{if } (n > 0 \wedge \text{hd}(x\ (n-1)) = 4] \text{ then } 1::\text{real} \text{ else } 0, \text{if } \text{hd}(x\ n) = 8 \text{ then } 1 \text{ else } 0])$ )))
proof -
  have  $\forall x\ n. ((\lambda x\ n. (([\text{if } (n > 0 \wedge \text{hd}(x\ (n-1)) = 4] \text{ then } 1::\text{real} \text{ else } 0,$ 
     $\text{if } (x\ n)!1 = 8 \text{ then } 1 \text{ else } 0]))) \text{ o } f\text{-Split2})\ x\ n$ 

```

```

    = (λx n. (([if (n > 0 ∧ hd(x (n-1)) = 4) then 1::real else 0,
                if hd(x n) = 8 then 1 else 0]))) x n
using f-Split2-def by simp
then show ?thesis
by metis
qed
then show ?thesis
using f6 f6-0 by auto
qed

```

Finally, *post-landing-finalize* is the composition of subsystems defined previously and other blocks. It is shown in *post-landing-finalize-1*.

```

abbreviation post-landing-finalize-part1 ≡ (
  (
    (
      (
        Split2 (* door-closed (boolean, 1/10s) is split into two *)
          ||B
          Id (* door-open-time: double *)
      ) ; ; Router 3 [0,2,1]
    )
    ||B
    post-mode
  )
  ||B
  (
    (UnitDelay 1.0 ; ; LopNOT) (* ac-on-ground *)
    ||B
    (UnitDelay 0) (* Delay2 *)
  )
)

```

```

abbreviation post-landing-finalize-part2 ≡ (
  (
    (LopNOT)
    ||B
    (Id) (* door-open-time: double *)
  ) ; ; variableTimer
)

```

```

abbreviation post-landing-finalize-part3 ≡ (
  (
    (LopAND 3)
    ||B
    (LopOR 2)
  ) ; ; latch
)

```

```

definition post-landing-finalize-1 ≡
(
  post-landing-finalize-part1 ; ;
  (
    post-landing-finalize-part2
    ||B

```

```

    post-landing-finalize-part3
  ) ;; LopAND 2 ;; rise1Shot ;; Split2
) fD (4, 1)

```

Simplified design corresponding to a part of the diagram from inputs to *variableTimer*.

abbreviation $plf\text{-}vt\text{-}simp \equiv \lambda x na. \text{if } (\text{if } hd(x na) = 0$
 $\text{then } (\text{if } na = 0 \text{ then } 0$
 $\text{else } \min (vT\text{-}fd\text{-}sol\text{-}1$
 $(\lambda n1. (\lambda na. \text{real-of-int}$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [Rate * \max (x na!(Suc 0)) 0]))) n1)$
 $(\lambda n1. (\text{if } hd(x n1) = 0 \text{ then } 1::\text{real else } 0)) (na - 1))$
 $((\lambda na. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [Rate * \max (x na!(Suc 0))$
 $0])))$
 $(na - 1))) + 1::\text{real}$
 $\text{else } 0) > (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [Rate * \max (x na!(Suc 0)) 0])))$
 $\text{then } 1::\text{real else } 0$

Simplified design corresponding to a part of the diagram from inputs to *latch*.

abbreviation $plf\text{-}latch\text{-}simp \equiv \lambda x na. (\text{latch-rec-calc-output}$
 $(\lambda n1. (\text{if } hd(x n1) = 0 \vee n1 = 0 \vee (x (n1-1))!2 \neq 4 \vee (x n1)!2 \neq 8$
 $\text{then } 0 \text{ else } 1::\text{real}))$
 $(\lambda n1. (\text{if } ((n1 = 0) \vee ((x (n1 - 1))!3 \neq 0 \wedge (x (n1 - 1))!4 = 0))$
 $\text{then } 0 \text{ else } 1::\text{real}))$
 $(na))$

A function for the simplified design corresponding to a part of the diagram from inputs to outputs but without the feedback from one of outputs.

abbreviation $plf\text{-}rise1shot\text{-}simp\text{-}f \equiv (\lambda x n. [\text{if } (((plf\text{-}vt\text{-}simp x n) \neq 0 \wedge (plf\text{-}latch\text{-}simp x n) \neq 0) \wedge$
 $(n > 0 \wedge ((plf\text{-}vt\text{-}simp x (n-1)) = 0 \vee (plf\text{-}latch\text{-}simp x (n-1)) = 0))) \text{ then } 1 \text{ else } 0,$
 $\text{if } (((plf\text{-}vt\text{-}simp x n) \neq 0 \wedge (plf\text{-}latch\text{-}simp x n) \neq 0) \wedge$
 $(n > 0 \wedge ((plf\text{-}vt\text{-}simp x (n-1)) = 0 \vee (plf\text{-}latch\text{-}simp x (n-1)) = 0))) \text{ then } 1 \text{ else } 0])$

Simplified design corresponding to a part of the diagram from inputs to outputs but without the feedback from one of outputs.

definition $plf\text{-}rise1shot\text{-}simp \equiv FBlock (\lambda x n. True) 5 2 plf\text{-}rise1shot\text{-}simp\text{-}f$

lemma *post-landing-finalize-1-simp-simblock:*

$post\text{-}landing\text{-}finalize\text{-}1 = plf\text{-}rise1shot\text{-}simp f_D (4, 1) \wedge SimBlock 5 2 plf\text{-}rise1shot\text{-}simp$

proof –

```

let ?f1-f = ( $\lambda x n. [hd(x n), hd(x n), hd(tl(x n))]$ )
let ?f1 = FBlock ( $\lambda x n. True$ ) 2 3 ?f1-f
have f1: Split2 (* door-closed (boolean, 1/10s) is split into two *)
  ||B Id (* door-open-time: double *)
  = FBlock ( $\lambda x n. True$ ) (1+1) (2+1)
  ( $\lambda x n. (((f\text{-}Split2 \circ (\lambda xx nn. take 1 (xx nn))) x n) \bullet$ 
    ( $(f\text{-}Id \circ (\lambda xx nn. drop 1 (xx nn)))) x n)$ )
using SimBlock-Id SimBlock-Split2 FBlock-parallel-comp
by (simp add: Split2-def simu-contract-real.Id-def)
then have f1-0: ... = ?f1
proof –

```

```

have  $\forall x n. ((\lambda x n. (((f\text{-}Split2 \circ (\lambda xx nn. take\ 1\ (xx\ nn)))\ x\ n)) \bullet$ 
   $((f\text{-}Id \circ (\lambda xx nn. drop\ 1\ (xx\ nn))))\ x\ n))\ x\ n)$ 
  = (?f1-f x n)
  using f-Id-def f-Split2-def by (simp add: drop-Suc hd-take-m)
then show ?thesis
  apply (simp)
  by (simp add: numeral-2-eq-2)
qed
have simblock-f1: SimBlock 2 3 (?f1)
  using SimBlock-Id SimBlock-Split2 SimBlock-FBlock-parallel-comp
  by (metis (no-types, lifting) One-nat-def Split2-def Suc-1 Suc-eq-plus1 f1 f1-0
    numeral-3-eq-3 simu-contract-real.Id-def)

let ?f2-f = ( $\lambda x n. [hd(x\ n), hd(tl(x\ n)), hd(x\ n)]$ )
let ?f2 = FBlock ( $\lambda x n. True$ ) (2) (3) ?f2-f
have f2: (Split2 ||B Id) ;; Router 3 [0,2,1] = ?f1 ;; Router 3 [0,2,1]
  using f1 f1-0 by auto
then have f2-0: ... = FBlock ( $\lambda x n. True$ ) (2) (3) (f-Router [0,2,1] o ?f1-f)
  using simblock-f1 Router-def SimBlock-Router FBlock-seq-comp by simp
then have f2-1: ... = ?f2
proof -
  have  $\forall x n. (f\text{-}Router\ [0,2,1]\ o\ ?f1\text{-}f)\ x\ n = ?f2\text{-}f\ x\ n$ 
    using f-Router-def by (simp)
  then show ?thesis
    by presburger
qed
have simblock-f2: SimBlock 2 3 ?f2
  using simblock-f1 SimBlock-Router SimBlock-FBlock-seq-comp
  by (metis (no-types, lifting) Router-def f2-0 f2-1 length-Cons list.size(3) numeral-3-eq-3)

let ?post-mode-f =
  ( $\lambda x n. ([if\ (n > 0 \wedge hd(x\ (n-1))) = 4\ then\ 1::real\ else\ 0,$ 
  if  $hd(x\ n) = 8\ then\ 1\ else\ 0])$ )
let ?post-mode = FBlock ( $\lambda x n. True$ ) (1) (2) ?post-mode-f
have simblock-post-mode: SimBlock 1 2 (?post-mode)
  apply (rule SimBlock-FBlock)
  apply (rule-tac x =  $\lambda na. [4]$  in exI)
  apply (rule-tac x =  $\lambda na. [if\ na > 0\ then\ 1\ else\ 0, 0]$  in exI)
  apply (simp add: f-blocks)
  by (simp add: f-blocks)
let ?f3-f = ( $\lambda x n. [hd(x\ n), hd(tl(x\ n)), hd(x\ n),$ 
  if  $(n > 0 \wedge (x\ (n-1))!2 = 4\ then\ 1::real\ else\ 0,$ 
  if  $(x\ n)!2 = 8\ then\ 1\ else\ 0]$ )
let ?f3 = FBlock ( $\lambda x n. True$ ) 3 5 ?f3-f
have f3: ((( Split2 (* door-closed (boolean, 1/10s) is split into two *)
  ||B
  Id (* door-open-time: double *)
  ) ;; Router 3 [0,2,1])
  ||B post-mode) = ?f2 ||B ?post-mode
  using f2 f2-0 f2-1 post-mode-simp by auto
then have f3-0: ... = FBlock ( $\lambda x n. True$ ) (2+1) (3+2)
  ( $\lambda x n. (((?f2-f \circ (\lambda xx nn. take\ 2\ (xx\ nn)))\ x\ n) \bullet$ 
   $((?post-mode-f \circ (\lambda xx nn. drop\ 2\ (xx\ nn))))\ x\ n))$ )
  using simblock-post-mode simblock-f1 FBlock-parallel-comp simblock-f2 by blast
then have f3-1: ... = FBlock ( $\lambda x n. True$ ) (2+1) (3+2) ?f3-f
proof -

```

```

show ?thesis
  apply (simp add: FBlock-def)
  apply (rel-simp)
  apply (rule conjI, clarify)
  apply (rule conjI, clarify)
  apply (rule iffI, clarify)
  defer
  apply (clarify)
  defer
  apply (clarify, rule iffI, clarify)
  apply (metis hd-drop-conv-nth lessI numeral-2-eq-2 numeral-3-eq-3)
  apply (clarify)
  apply (simp add: hd-drop-conv-nth)
  apply (clarify, rule conjI, clarify)
  apply (rule iffI, clarify)
  apply (metis hd-drop-conv-nth lessI numeral-2-eq-2 numeral-3-eq-3)
  apply (clarify)
  apply (simp add: hd-drop-conv-nth)
  apply (clarify, rule iffI, clarify)
  defer
  apply (clarify)
  defer
  proof -
    fix  $ok_v$   $ok_v'::bool$  and  $inouts_v$   $inouts_v'::nat \Rightarrow real\ list$  and  $x$ 
    assume  $a1: \forall x. (hd (drop\ 2\ (inouts_v\ x)) = 8 \longrightarrow$ 
       $(0 < x \wedge hd (drop\ 2\ (inouts_v\ (x - Suc\ 0))) = 4 \longrightarrow$ 
         $length(inouts_v\ x) = 3 \wedge$ 
         $length(inouts_v'\ x) = 5 \wedge$ 
         $[hd (take\ 2\ (inouts_v\ x)), hd (tl (take\ 2\ (inouts_v\ x))), hd (take\ 2\ (inouts_v\ x)), 1, 1] =$ 
         $inouts_v'\ x) \wedge$ 
         $(x = 0 \longrightarrow$ 
           $length(inouts_v\ 0) = 3 \wedge$ 
           $length(inouts_v'\ 0) = 5 \wedge$ 
           $[hd (take\ 2\ (inouts_v\ 0)), hd (tl (take\ 2\ (inouts_v\ 0))), hd (take\ 2\ (inouts_v\ 0)), 0, 1] =$ 
           $inouts_v'\ 0) \wedge$ 
           $(\neg hd (drop\ 2\ (inouts_v\ (x - Suc\ 0))) = 4 \longrightarrow$ 
             $length(inouts_v\ x) = 3 \wedge$ 
             $length(inouts_v'\ x) = 5 \wedge$ 
             $[hd (take\ 2\ (inouts_v\ x)), hd (tl (take\ 2\ (inouts_v\ x))), hd (take\ 2\ (inouts_v\ x)), 0, 1] =$ 
             $inouts_v'\ x) \wedge$ 
             $(\neg hd (drop\ 2\ (inouts_v\ x)) = 8 \longrightarrow$ 
               $(0 < x \wedge hd (drop\ 2\ (inouts_v\ (x - Suc\ 0))) = 4 \longrightarrow$ 
                 $length(inouts_v\ x) = 3 \wedge$ 
                 $length(inouts_v'\ x) = 5 \wedge$ 
                 $[hd (take\ 2\ (inouts_v\ x)), hd (tl (take\ 2\ (inouts_v\ x))), hd (take\ 2\ (inouts_v\ x)), 1, 0] =$ 
                 $inouts_v'\ x) \wedge$ 
                 $(x = 0 \longrightarrow$ 
                   $length(inouts_v\ 0) = 3 \wedge$ 
                   $length(inouts_v'\ 0) = 5 \wedge$ 
                   $[hd (take\ 2\ (inouts_v\ 0)), hd (tl (take\ 2\ (inouts_v\ 0))), hd (take\ 2\ (inouts_v\ 0)), 0, 1] =$ 
                   $inouts_v'\ 0) \wedge$ 
                   $(\neg hd (drop\ 2\ (inouts_v\ (x - Suc\ 0))) = 4 \longrightarrow$ 
                     $length(inouts_v\ x) = 3 \wedge$ 
                     $length(inouts_v'\ x) = 5 \wedge$ 
                     $[hd (take\ 2\ (inouts_v\ x)), hd (tl (take\ 2\ (inouts_v\ x))), hd (take\ 2\ (inouts_v\ x)), 0, 0] =$ 

```



```

    inouts_v' x))
from a1 have len-3:  $\forall x. \text{length}(\text{inouts}_v x) = 3$ 
  by (metis neg0-conv)
have drop-2:  $\forall x. (\text{hd} (\text{drop } 2 (\text{inouts}_v' x)) = (\text{inouts}_v' x)!2)$ 
  using len-3 hd-drop-m
  by (metis Suc-eq-plus1 Suc-le-eq a1 add-Suc-right add-diff-cancel-right' diff-le-self
    hd-drop-conv-nth neg0-conv one-plus-numeral one-plus-numeral-commute semiring-norm(2)
      semiring-norm(3) semiring-norm(4))
have take-2:  $\forall x. \text{hd} (\text{take } 2 (\text{inouts}_v x)) = \text{hd}(\text{inouts}_v x)$ 
  using len-3 hd-take-m by simp
have take-tl-2:  $\forall x. \text{hd} (\text{tl} (\text{take } 2 (\text{inouts}_v x))) = \text{hd}(\text{tl}(\text{inouts}_v x))$ 
  using len-3 hd-tl-take-m by simp
show (inouts_v x)!2 = 8  $\longrightarrow$ 
  (0 < x  $\wedge$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
    length(inouts_v x) = 3  $\wedge$ 
    length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1] =
inouts_v' x)  $\wedge$ 
  (x = 0  $\longrightarrow$ 
    length(inouts_v 0) = 3  $\wedge$ 
    length(inouts_v' 0) = 5  $\wedge$  [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 1] =
inouts_v' 0)  $\wedge$ 
  ( $\neg$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
    length(inouts_v x) = 3  $\wedge$ 
    length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1] =
inouts_v' x)  $\wedge$ 
  ( $\neg$  inouts_v x!2 = 8  $\longrightarrow$ 
    (0 < x  $\wedge$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
      length(inouts_v x) = 3  $\wedge$ 
      length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0] =
inouts_v' x)  $\wedge$ 
    (x = 0  $\longrightarrow$ 
      length(inouts_v 0) = 3  $\wedge$ 
      length(inouts_v' 0) = 5  $\wedge$  [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 1] =
inouts_v' 0)  $\wedge$ 
    ( $\neg$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
      length(inouts_v x) = 3  $\wedge$ 
      length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0] =
inouts_v' x)
  using drop-2 take-2 take-tl-2
  by (metis One-nat-def Suc-1 a1 hd-drop-conv-nth len-3 lessI numeral-3-eq-3)
next
fix ok_v ok_v':bool and inouts_v inouts_v':nat  $\Rightarrow$  real list and x
assume a1:  $\forall x. (\text{inouts}_v x)!2 = 8 \longrightarrow$ 
  (0 < x  $\wedge$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
    length(inouts_v x) = 3  $\wedge$ 
    length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1] =
inouts_v' x)  $\wedge$ 
  (x = 0  $\longrightarrow$ 
    length(inouts_v 0) = 3  $\wedge$ 
    length(inouts_v' 0) = 5  $\wedge$  [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 1] =
inouts_v' 0)  $\wedge$ 
  ( $\neg$  inouts_v (x - Suc 0)!2 = 4  $\longrightarrow$ 
    length(inouts_v x) = 3  $\wedge$ 
    length(inouts_v' x) = 5  $\wedge$  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1] =

```

$inouts_v' x)) \wedge$
 $(\neg inouts_v x!(2) = 8 \longrightarrow$
 $(0 < x \wedge inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$
 $length(inouts_v' x) = 5 \wedge [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0] =$
 $inouts_v' x) \wedge$
 $(x = 0 \longrightarrow$
 $length(inouts_v 0) = 3 \wedge$
 $length(inouts_v' 0) = 5 \wedge [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 1] =$
 $inouts_v' 0) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$
 $length(inouts_v' x) = 5 \wedge [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0] =$
 $inouts_v' x))$
from $a1$ **have** $len-3: \forall x. length(inouts_v x) = 3$
by (*metis neq0-conv*)
have $drop-2: \forall x. (hd (drop 2 (inouts_v' x)) = (inouts_v' x)!(2))$
using $len-3$ $hd-drop-m$
by (*metis Suc-eq-plus1 Suc-le-eq a1 add-Suc-right add-diff-cancel-right' diff-le-self*
 $hd-drop-conv-nth neq0-conv one-plus-numeral one-plus-numeral-commute semiring-norm(2)$
 $semiring-norm(3) semiring-norm(4))$
have $take-2: \forall x. hd (take 2 (inouts_v x)) = hd(inouts_v x)$
using $len-3$ $hd-take-m$ **by** *simp*
have $take-tl-2: \forall x. hd (tl (take 2 (inouts_v x))) = hd(tl(inouts_v x))$
using $len-3$ $hd-tl-take-m$ **by** *simp*
show $(hd (drop 2 (inouts_v x)) = 8 \longrightarrow$
 $(0 < x \wedge hd (drop 2 (inouts_v (x - Suc 0)))) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$
 $length(inouts_v' x) = 5 \wedge$
 $[hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2 (inouts_v x)), 1, 1] =$
 $inouts_v' x) \wedge$
 $(x = 0 \longrightarrow$
 $length(inouts_v 0) = 3 \wedge$
 $length(inouts_v' 0) = 5 \wedge$
 $[hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2 (inouts_v 0)), 0, 1] =$
 $inouts_v' 0) \wedge$
 $(\neg hd (drop 2 (inouts_v (x - Suc 0)))) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$
 $length(inouts_v' x) = 5 \wedge$
 $[hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2 (inouts_v x)), 0, 1] =$
 $inouts_v' x) \wedge$
 $(\neg hd (drop 2 (inouts_v x)) = 8 \longrightarrow$
 $(0 < x \wedge hd (drop 2 (inouts_v (x - Suc 0)))) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$
 $length(inouts_v' x) = 5 \wedge$
 $[hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2 (inouts_v x)), 1, 0] =$
 $inouts_v' x) \wedge$
 $(x = 0 \longrightarrow$
 $length(inouts_v 0) = 3 \wedge$
 $length(inouts_v' 0) = 5 \wedge$
 $[hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2 (inouts_v 0)), 0, 1] =$
 $inouts_v' 0) \wedge$
 $(\neg hd (drop 2 (inouts_v (x - Suc 0)))) = 4 \longrightarrow$
 $length(inouts_v x) = 3 \wedge$

```

    length(inouts_v' x) = 5 ∧
    [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2 (inouts_v x)), 0, 0] =
    inouts_v' x)
  using drop-2 take-2 take-tl-2
  by (metis One-nat-def Suc-1 a1 hd-drop-conv-nth len-3 lessI numeral-3-eq-3)
next
fix ok_v ok_v'::bool and inouts_v inouts_v'::nat ⇒ real list and x::nat
assume a1: ∀x. (hd (drop 2 (inouts_v x)) = 8 →
  (0 < x ∧ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
    length(inouts_v x) = 3 ∧
    length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 1, 1] = inouts_v' x) ∧
    (x = 0 →
      length(inouts_v 0) = 3 ∧
      length(inouts_v' 0) = 5 ∧ [hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2
(inouts_v 0)), 0, 0] = inouts_v' 0) ∧
      (¬ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
        length(inouts_v x) = 3 ∧
        length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 0, 1] = inouts_v' x) ∧
        (¬ hd (drop 2 (inouts_v x)) = 8 →
          (0 < x ∧ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
            length(inouts_v x) = 3 ∧
            length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 1, 0] = inouts_v' x) ∧
            (x = 0 →
              length(inouts_v 0) = 3 ∧
              length(inouts_v' 0) = 5 ∧ [hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2
(inouts_v 0)), 0, 0] = inouts_v' 0) ∧
              (¬ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
                length(inouts_v x) = 3 ∧
                length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 0, 0] = inouts_v' x))
          assume a2: ¬ hd (drop 2 (inouts_v 0)) = 8
          assume a3: ¬ inouts_v 0!(2) = 8
          from a1 have len-3: ∀x. length(inouts_v x) = 3
          by (metis neq0-conv)
          have drop-2: ∀x. (hd (drop 2 (inouts_v' x)) = (inouts_v' x)!2)
          using len-3 hd-drop-m
          by (metis Suc-eq-plus1 Suc-le-eq a1 add-Suc-right add-diff-cancel-right' diff-le-self
            hd-drop-conv-nth neq0-conv one-plus-numeral one-plus-numeral-commute semiring-norm(2)
              semiring-norm(3) semiring-norm(4))
          have take-2: ∀x. hd (take 2 (inouts_v x)) = hd(inouts_v x)
          using len-3 hd-take-m by simp
          have take-tl-2: ∀x. hd (tl (take 2 (inouts_v x))) = hd(tl(inouts_v x))
          using len-3 hd-tl-take-m by simp
          show (inouts_v x)!(2) = 8 →
            (0 < x ∧ inouts_v (x - Suc 0)!(2) = 4 →
              length(inouts_v x) = 3 ∧
              length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1] =
inouts_v' x) ∧
            (x = 0 →
              length(inouts_v 0) = 3 ∧
              length(inouts_v' 0) = 5 ∧ [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 0] =

```

```

inouts_v' 0) ∧
  (¬ inouts_v (x - Suc 0)!(2) = 4 →
    length(inouts_v x) = 3 ∧
    length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1] =
inouts_v' x) ∧
  (¬ inouts_v x!(2) = 8 →
    (0 < x ∧ inouts_v (x - Suc 0)!(2) = 4 →
      length(inouts_v x) = 3 ∧
      length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0] =
inouts_v' x) ∧
    (x = 0 →
      length(inouts_v 0) = 3 ∧
      length(inouts_v' 0) = 5 ∧ [hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 0] =
inouts_v' 0) ∧
    (¬ inouts_v (x - Suc 0)!(2) = 4 →
      length(inouts_v x) = 3 ∧
      length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0] =
inouts_v' x))
  using drop-2 take-2 take-tl-2
  by (metis One-nat-def Suc-1 a1 hd-drop-conv-nth len-3 lessI numeral-3-eq-3)
next
fix ok_v ok_v'::bool and inouts_v inouts_v'::nat ⇒ real list and x
assume a1: ∀x. (inouts_v x!(2) = 8 →
  (0 < x ∧ inouts_v (x - Suc 0)!(2) = 4 →
    length(inouts_v x) = 3 ∧ length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd
(inouts_v x), 1, 1] = inouts_v' x) ∧
    (x = 0 →
      length(inouts_v 0) = 3 ∧ length(inouts_v' 0) = 5 ∧ [hd (inouts_v 0), hd (tl (inouts_v 0)), hd
(inouts_v 0), 0, 0] = inouts_v' 0) ∧
      (¬ inouts_v (x - Suc 0)!(2) = 4 →
        length(inouts_v x) = 3 ∧ length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd
(inouts_v x), 0, 1] = inouts_v' x)) ∧
        (¬ inouts_v x!(2) = 8 →
          (0 < x ∧ inouts_v (x - Suc 0)!(2) = 4 →
            length(inouts_v x) = 3 ∧ length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd
(inouts_v x), 1, 0] = inouts_v' x) ∧
            (x = 0 →
              length(inouts_v 0) = 3 ∧ length(inouts_v' 0) = 5 ∧ [hd (inouts_v 0), hd (tl (inouts_v 0)), hd
(inouts_v 0), 0, 0] = inouts_v' 0) ∧
              (¬ inouts_v (x - Suc 0)!(2) = 4 →
                length(inouts_v x) = 3 ∧ length(inouts_v' x) = 5 ∧ [hd (inouts_v x), hd (tl (inouts_v x)), hd
(inouts_v x), 0, 0] = inouts_v' x))
                from a1 have len-3: ∀x. length(inouts_v x) = 3
                by (metis neq0-conv)
                have drop-2: ∀x. (hd (drop 2 (inouts_v' x)) = (inouts_v' x)!)2)
                using len-3 hd-drop-m
                by (metis Suc-eq-plus1 Suc-le-eq a1 add-Suc-right add-diff-cancel-right' diff-le-self
                    hd-drop-conv-nth neq0-conv one-plus-numeral one-plus-numeral-commute semiring-norm(2)
                        semiring-norm(3) semiring-norm(4))
                have take-2: ∀x. hd (take 2 (inouts_v x)) = hd(inouts_v x)
                using len-3 hd-take-m by simp
                have take-tl-2: ∀x. hd (tl (take 2 (inouts_v x))) = hd(tl(inouts_v x))
                using len-3 hd-tl-take-m by simp
                show (hd (drop 2 (inouts_v x)) = 8 →

```

```

(0 < x ∧ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
  length(inouts_v x) = 3 ∧
  length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 1, 1] = inouts_v' x) ∧
  (x = 0 →
    length(inouts_v 0) = 3 ∧
    length(inouts_v' 0) = 5 ∧ [hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2
(inouts_v 0)), 0, 0] = inouts_v' 0) ∧
    (¬ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
      length(inouts_v x) = 3 ∧
      length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 0, 1] = inouts_v' x) ∧
      (¬ hd (drop 2 (inouts_v x)) = 8 →
        (0 < x ∧ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
          length(inouts_v x) = 3 ∧
          length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 1, 0] = inouts_v' x) ∧
          (x = 0 →
            length(inouts_v 0) = 3 ∧
            length(inouts_v' 0) = 5 ∧ [hd (take 2 (inouts_v 0)), hd (tl (take 2 (inouts_v 0))), hd (take 2
(inouts_v 0)), 0, 0] = inouts_v' 0) ∧
            (¬ hd (drop 2 (inouts_v (x - Suc 0))) = 4 →
              length(inouts_v x) = 3 ∧
              length(inouts_v' x) = 5 ∧ [hd (take 2 (inouts_v x)), hd (tl (take 2 (inouts_v x))), hd (take 2
(inouts_v x)), 0, 0] = inouts_v' x))
          using drop-2 take-2 take-tl-2
          by (metis One-nat-def Suc-1 a1 hd-drop-conv-nth len-3 lessI numeral-3-eq-3)
        qed
      qed
    )
  )
have simblock-f3: SimBlock 3 5 (?f3)
  using simblock-f2 simblock-post-mode SimBlock-FBlock-parallel-comp
  by (smt Suc-eq-plus1 add-Suc f3-0 f3-1 numeral-2-eq-2 numeral-3-eq-3 numeral-code(3))

let ?f4-f = (λx n. [(if n = 0 then 0 else (if hd(x (n-1)) = 0 then 1 else 0)])
let ?f4 = FBlock (λx n. True) 1 1 ?f4-f
have f4: (UnitDelay 1.0 ;; LopNOT) = FBlock (λx n. True) 1 1 (f-LopNOT o f-UnitDelay 1.0)
using SimBlock-UnitDelay SimBlock-LopNOT FBlock-seq-comp by (simp add: LopNOT-def UnitDelay-def)
then have f4-0: ... = FBlock (λx n. True) 1 1 ?f4-f
proof -
  have ∀ x n. (f-LopNOT o f-UnitDelay 1.0) x n = ?f4-f x n
  using f-LopNOT-def f-UnitDelay-def by simp
  then show ?thesis
  by presburger
qed
have simblock-f4: SimBlock 1 1 ?f4
  using SimBlock-UnitDelay SimBlock-LopNOT SimBlock-FBlock-seq-comp
  by (metis (no-types, lifting) LopNOT-def UnitDelay-def f4 f4-0)

let ?f5-f = (λx n. [(if n = 0 then 0 else (if hd(x (n-1)) = 0 then 1 else 0),
  if n = 0 then 0 else hd(tl(x (n - 1)))])
let ?f5 = FBlock (λx n. True) 2 2 ?f5-f
have f5: ((UnitDelay 1.0 ;; LopNOT)
  ||B
  (UnitDelay 0) (* Delay2 *))
= ?f4 ||B (UnitDelay 0)

```

```

using f4 f4-0 by auto
then have f5-0: ... = FBlock (λx n. True) 2 2
  (λx n. (((?f4-f ◦ (λxx nn. take 1 (xx nn))) x n) •
    ((f-UnitDelay 0 ◦ (λxx nn. drop 1 (xx nn)))) x n))
using simblock-f4 SimBlock-UnitDelay FBlock-parallel-comp apply (simp add: UnitDelay-def)
by (simp add: numeral-2-eq-2)
then have f5-1: ... = ?f5
proof –
  have ∀ x n. (λx n. (((?f4-f ◦ (λxx nn. take 1 (xx nn))) x n) •
    ((f-UnitDelay 0 ◦ (λxx nn. drop 1 (xx nn)))) x n)) x n
    = ?f5-f x n
  using f-UnitDelay-def apply (simp)
  apply (rule allI)+
  apply (rule conjI, clarify)
  apply (simp add: drop-Suc hd-take-m)
  by (simp add: drop-Suc hd-take-m)
  then show ?thesis
    by presburger
qed
have simblock-f5: SimBlock 2 2 ?f5
  using simblock-f4 SimBlock-UnitDelay SimBlock-FBlock-parallel-comp f5 f5-0 f5-1
  by (metis (no-types, lifting) Suc-1 Suc-eq-plus1 UnitDelay-def)

let ?f6-f = (λx n. [hd(x n), hd(tl(x n)), hd(x n),
  if (n > 0 ∧ (x (n-1))!2 = 4) then 1::real else 0,
  if (x n)!2 = 8 then 1 else 0,
  (if n = 0 then 0 else (if (x (n - 1))!3 = 0 then 1 else 0)),
  if n = 0 then 0 else (x (n - 1))!4])
let ?f6 = FBlock (λx n. True) 5 7 ?f6-f
have f6: (((
  Split2 (* door-closed (boolean, 1/10s) is split into two *)
  ||B
  Id (* door-open-time: double *)
  ) ; ; Router 3 [0,2,1])
  ||B
  post-mode
  )
  ||B
  (
  (UnitDelay 1.0 ; ; LopNOT)
  ||B
  (UnitDelay 0) (* Delay2 *)
  ))
  = ?f3 ||B ?f5
by (smt Suc3-eq-add-3 Suc-eq-plus1 add-2-eq-Suc eval-nat-numeral(3) f1 f1-0 f2-0 f2-1 f3-0
  f3-1 f4 f4-0 f5-0 f5-1 numeral-Bit0 post-mode-simp)
then have f6-0: ... = FBlock (λx n. True) (3 + 2) (5 + 2)
  (λx n. (((?f3-f ◦ (λxx nn. take 3 (xx nn))) x n) •
    ((?f5-f ◦ (λxx nn. drop 3 (xx nn)))) x n))
using simblock-f3 simblock-f5 FBlock-parallel-comp by (simp)
then have f6-1: ... = FBlock (λx n. True) (3 + 2) (5 + 2) ?f6-f
proof –
  show ?thesis
    apply (simp add: FBlock-def)
    apply (rel-simp)

```

```

apply (rule conjI, clarify, rule iffI)
apply (clarify)
defer
apply (clarify)
defer
apply (clarify, rule iffI)
apply (clarify)
defer
apply (clarify)
defer
proof -
  fix  $ok_v$  and  $inouts_v::nat \Rightarrow real$  list and  $ok_v'$  and  $inouts_v':::nat \Rightarrow real$  list and  $x::nat$ 
  assume  $a1: \forall x. (x = 0 \longrightarrow$ 
     $length(inouts_v\ 0) = 5 \wedge$ 
     $length(inouts_v'\ 0) = 7 \wedge$ 
     $[hd\ (take\ 3\ (inouts_v\ 0)),\ hd\ (tl\ (take\ 3\ (inouts_v\ 0))),\ hd\ (take\ 3\ (inouts_v\ 0)),\ 0,\ 1,\ 0,\ 0] =$ 
     $inouts_v'\ 0) \wedge$ 
     $(0 < x \longrightarrow$ 
       $(hd\ (drop\ 3\ (inouts_v\ (x - Suc\ 0)))) = 0 \longrightarrow$ 
       $(inouts_v\ x!(2) = 8 \longrightarrow$ 
         $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
           $length(inouts_v\ x) = 5 \wedge$ 
           $length(inouts_v'\ x) = 7 \wedge$ 
           $[hd\ (take\ 3\ (inouts_v\ x)),\ hd\ (tl\ (take\ 3\ (inouts_v\ x))),\ hd\ (take\ 3\ (inouts_v\ x)),\ 1,\ 1,\ 1,$ 
             $hd\ (tl\ (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$ 
             $inouts_v'\ x) \wedge$ 
           $(\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
             $length(inouts_v\ x) = 5 \wedge$ 
             $length(inouts_v'\ x) = 7 \wedge$ 
             $[hd\ (take\ 3\ (inouts_v\ x)),\ hd\ (tl\ (take\ 3\ (inouts_v\ x))),\ hd\ (take\ 3\ (inouts_v\ x)),\ 0,\ 1,\ 1,$ 
               $hd\ (tl\ (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$ 
               $inouts_v'\ x) \wedge$ 
             $(\neg inouts_v\ x!(2) = 8 \longrightarrow$ 
               $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
                 $length(inouts_v\ x) = 5 \wedge$ 
                 $length(inouts_v'\ x) = 7 \wedge$ 
                 $[hd\ (take\ 3\ (inouts_v\ x)),\ hd\ (tl\ (take\ 3\ (inouts_v\ x))),\ hd\ (take\ 3\ (inouts_v\ x)),\ 1,\ 0,\ 1,$ 
                   $hd\ (tl\ (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$ 
                   $inouts_v'\ x) \wedge$ 
                 $(\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
                   $length(inouts_v\ x) = 5 \wedge$ 
                   $length(inouts_v'\ x) = 7 \wedge$ 
                   $[hd\ (take\ 3\ (inouts_v\ x)),\ hd\ (tl\ (take\ 3\ (inouts_v\ x))),\ hd\ (take\ 3\ (inouts_v\ x)),\ 0,\ 0,\ 1,$ 
                     $hd\ (tl\ (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$ 
                     $inouts_v'\ x) \wedge$ 
                   $(\neg hd\ (drop\ 3\ (inouts_v\ (x - Suc\ 0)))) = 0 \longrightarrow$ 
                     $(inouts_v\ x!(2) = 8 \longrightarrow$ 
                       $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
                         $length(inouts_v\ x) = 5 \wedge$ 
                         $length(inouts_v'\ x) = 7 \wedge$ 
                         $[hd\ (take\ 3\ (inouts_v\ x)),\ hd\ (tl\ (take\ 3\ (inouts_v\ x))),\ hd\ (take\ 3\ (inouts_v\ x)),\ 1,\ 1,\ 0,$ 
                           $hd\ (tl\ (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$ 
                           $inouts_v'\ x) \wedge$ 
                         $(\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$ 
                           $length(inouts_v\ x) = 5 \wedge$ 

```

$length(inouts_v' x) = 7 \wedge$
 $[hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 1, 0,$
 $hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v\ x!(2) = 8 \longrightarrow$
 $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 1, 0, 0,$
 $hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 0, 0,$
 $hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] =$
 $inouts_v' x))$

from $a1$ **have** $len-5: \forall x. length(inouts_v\ x) = 5$
by (*metis neq0-conv*)

have $hd-take-3: hd (take\ 3\ (inouts_v\ x)) = hd(inouts_v\ x)$
using $len-5$ **by** (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral*)

have $hd-tl-take-3: hd (tl (take\ 3\ (inouts_v\ x))) = hd (tl (inouts_v\ x))$
using $len-5$ **by** (*simp add: hd-tl-take-m*)

have $hd-drop-3: hd (drop\ 3\ (inouts_v\ x)) = inouts_v\ x!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)

have $hd-drop-3': hd (drop\ 3\ (inouts_v\ (x - Suc\ 0))) = inouts_v\ (x - Suc\ 0)!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)

have $hd-tl-drop-3: hd (tl (drop\ 3\ (inouts_v\ x))) = inouts_v\ x!(4)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth nth-tl tl-drop*)

have $hd-tl-drop-3': hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0)))) = inouts_v\ (x - Suc\ 0)!(4)$
using $len-5$
by (*metis drop-Suc eval-nat-numeral(2) eval-nat-numeral(3) hd-drop-conv-nth lessI*
semiring-norm(26) semiring-norm(27) tl-drop)

show $(x = 0 \longrightarrow$
 $length(inouts_v\ 0) = 5 \wedge$
 $length(inouts_v' 0) = 7 \wedge$
 $[hd (inouts_v\ 0), hd (tl (inouts_v\ 0)), hd (inouts_v\ 0), 0, 1, 0, 0] = inouts_v' 0) \wedge$
 $(0 < x \longrightarrow$
 $(inouts_v\ (x - Suc\ 0)!(3) = 0 \longrightarrow$
 $(inouts_v\ x!(2) = 8 \longrightarrow$
 $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v\ x), hd (tl (inouts_v\ x)), hd (inouts_v\ x), 1, 1, 1, inouts_v\ (x - Suc\ 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v\ x), hd (tl (inouts_v\ x)), hd (inouts_v\ x), 0, 1, 1, inouts_v\ (x - Suc\ 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v\ x!(2) = 8 \longrightarrow$
 $(inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v\ x), hd (tl (inouts_v\ x)), hd (inouts_v\ x), 1, 0, 1, inouts_v\ (x - Suc\ 0)!(4)] =$


```

    inouts_v' x) ∧
    (¬ inouts_v (x - Suc 0)!(2) = 4 →
    length(inouts_v x) = 5 ∧
    length(inouts_v' x) = 7 ∧
    [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0, 1, inouts_v (x - Suc 0)!(4)] =
    inouts_v' x)) ∧
  (¬ inouts_v (x - Suc 0)!(3) = 0 →
  (inouts_v x!(2) = 8 →
  (inouts_v (x - Suc 0)!(2) = 4 →
  length(inouts_v x) = 5 ∧
  length(inouts_v' x) = 7 ∧
  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1, 0, inouts_v (x - Suc 0)!(4)] =
  inouts_v' x) ∧
  (¬ inouts_v (x - Suc 0)!(2) = 4 →
  length(inouts_v x) = 5 ∧
  length(inouts_v' x) = 7 ∧
  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1, 0, inouts_v (x - Suc 0)!(4)] =
  inouts_v' x) ∧
  (¬ inouts_v x!(2) = 8 →
  (inouts_v (x - Suc 0)!(2) = 4 →
  length(inouts_v x) = 5 ∧
  length(inouts_v' x) = 7 ∧
  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0, 0, inouts_v (x - Suc 0)!(4)] =
  inouts_v' x) ∧
  (¬ inouts_v (x - Suc 0)!(2) = 4 →
  length(inouts_v x) = 5 ∧
  length(inouts_v' x) = 7 ∧
  [hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0, 0, inouts_v (x - Suc 0)!(4)] =
  inouts_v' x))))
  using a1 hd-take-3 hd-tl-take-3 hd-drop-3' hd-tl-drop-3' by (smt )
next
fix ok_v and inouts_v::nat⇒real list and ok_v' and inouts_v'::nat⇒real list and x::nat
assume a1: ∀x. (x = 0 →
length(inouts_v 0) = 5 ∧
length(inouts_v' 0) = 7 ∧
[hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 1, 0, 0] = inouts_v' 0) ∧
(0 < x →
(inouts_v (x - Suc 0)!(3) = 0 →
(inouts_v x!(2) = 8 →
(inouts_v (x - Suc 0)!(2) = 4 →
length(inouts_v x) = 5 ∧
length(inouts_v' x) = 7 ∧
[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1, 1, inouts_v (x - Suc 0)!(4)] =
inouts_v' x) ∧
(¬ inouts_v (x - Suc 0)!(2) = 4 →
length(inouts_v x) = 5 ∧
length(inouts_v' x) = 7 ∧
[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1, 1, inouts_v (x - Suc 0)!(4)] =
inouts_v' x) ∧
(¬ inouts_v x!(2) = 8 →
(inouts_v (x - Suc 0)!(2) = 4 →
length(inouts_v x) = 5 ∧
length(inouts_v' x) = 7 ∧
[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0, 1, inouts_v (x - Suc 0)!(4)] =
inouts_v' x) ∧

```

$(\neg \text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$
 $[\text{hd}(\text{inouts}_v x), \text{hd}(\text{tl}(\text{inouts}_v x)), \text{hd}(\text{inouts}_v x), 0, 0, 1, \text{inouts}_v (x - \text{Suc } 0)!(4)] =$
 $\text{inouts}_v' x)) \wedge$
 $(\neg \text{inouts}_v (x - \text{Suc } 0)!(3) = 0 \longrightarrow$
 $(\text{inouts}_v x!(2) = 8 \longrightarrow$
 $(\text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$
 $[\text{hd}(\text{inouts}_v x), \text{hd}(\text{tl}(\text{inouts}_v x)), \text{hd}(\text{inouts}_v x), 1, 1, 0, \text{inouts}_v (x - \text{Suc } 0)!(4)] =$
 $\text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$
 $[\text{hd}(\text{inouts}_v x), \text{hd}(\text{tl}(\text{inouts}_v x)), \text{hd}(\text{inouts}_v x), 0, 1, 0, \text{inouts}_v (x - \text{Suc } 0)!(4)] =$
 $\text{inouts}_v' x)) \wedge$
 $(\neg \text{inouts}_v x!(2) = 8 \longrightarrow$
 $(\text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$
 $[\text{hd}(\text{inouts}_v x), \text{hd}(\text{tl}(\text{inouts}_v x)), \text{hd}(\text{inouts}_v x), 1, 0, 0, \text{inouts}_v (x - \text{Suc } 0)!(4)] =$
 $\text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$
 $[\text{hd}(\text{inouts}_v x), \text{hd}(\text{tl}(\text{inouts}_v x)), \text{hd}(\text{inouts}_v x), 0, 0, 0, \text{inouts}_v (x - \text{Suc } 0)!(4)] =$
 $\text{inouts}_v' x))))$

from $a1$ **have** $\text{len-5}: \forall x. \text{length}(\text{inouts}_v x) = 5$
by (*metis neq0-conv*)
have $\text{hd-take-3}: \text{hd}(\text{take } 3 (\text{inouts}_v x)) = \text{hd}(\text{inouts}_v x)$
using len-5 **by** (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral*)
have $\text{hd-tl-take-3}: \text{hd}(\text{tl}(\text{take } 3 (\text{inouts}_v x))) = \text{hd}(\text{tl}(\text{inouts}_v x))$
using len-5 **by** (*simp add: hd-tl-take-m*)
have $\text{hd-drop-3}: \text{hd}(\text{drop } 3 (\text{inouts}_v x)) = \text{inouts}_v x!(3)$
using len-5 **by** (*simp add: hd-drop-conv-nth*)
have $\text{hd-drop-3}': \text{hd}(\text{drop } 3 (\text{inouts}_v (x - \text{Suc } 0))) = \text{inouts}_v (x - \text{Suc } 0)!(3)$
using len-5 **by** (*simp add: hd-drop-conv-nth*)
have $\text{hd-tl-drop-3}: \text{hd}(\text{tl}(\text{drop } 3 (\text{inouts}_v x))) = \text{inouts}_v x!(4)$
using len-5 **by** (*simp add: hd-drop-conv-nth nth-tl tl-drop*)
have $\text{hd-tl-drop-3}': \text{hd}(\text{tl}(\text{drop } 3 (\text{inouts}_v (x - \text{Suc } 0)))) = \text{inouts}_v (x - \text{Suc } 0)!(4)$
using len-5
by (*metis drop-Suc eval-nat-numeral(2) eval-nat-numeral(3) hd-drop-conv-nth lessI*
semiring-norm(26) semiring-norm(27) tl-drop)

show $(x = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v 0) = 5 \wedge$
 $\text{length}(\text{inouts}_v' 0) = 7 \wedge$
 $[\text{hd}(\text{take } 3 (\text{inouts}_v 0)), \text{hd}(\text{tl}(\text{take } 3 (\text{inouts}_v 0))), \text{hd}(\text{take } 3 (\text{inouts}_v 0)), 0, 1, 0, 0] =$
 $\text{inouts}_v' 0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd}(\text{drop } 3 (\text{inouts}_v (x - \text{Suc } 0))) = 0 \longrightarrow$
 $(\text{inouts}_v x!(2) = 8 \longrightarrow$
 $(\text{inouts}_v (x - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 7 \wedge$

$$\begin{aligned}
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 1, 1, 1, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x \wedge \\
& (\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 1, 1, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x) \wedge \\
& (\neg inouts_v\ x!(2) = 8 \longrightarrow \\
& (inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 1, 0, 1, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x) \wedge \\
& (\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 0, 1, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x)) \wedge \\
& (\neg hd (drop\ 3\ (inouts_v\ (x - Suc\ 0))) = 0 \longrightarrow \\
& (inouts_v\ x!(2) = 8 \longrightarrow \\
& (inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 1, 1, 0, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x) \wedge \\
& (\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 1, 0, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x) \wedge \\
& (\neg inouts_v\ x!(2) = 8 \longrightarrow \\
& (inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 1, 0, 0, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x) \wedge \\
& (\neg inouts_v\ (x - Suc\ 0)!(2) = 4 \longrightarrow \\
& \quad length(inouts_v\ x) = 5 \wedge \\
& \quad length(inouts_v'\ x) = 7 \wedge \\
& [hd (take\ 3\ (inouts_v\ x)), hd (tl (take\ 3\ (inouts_v\ x))), hd (take\ 3\ (inouts_v\ x)), 0, 0, 0, \\
& \quad hd (tl (drop\ 3\ (inouts_v\ (x - Suc\ 0))))] = \\
& inouts_v'\ x))))
\end{aligned}$$

using a1 hd-take-3 hd-tl-take-3 hd-drop-3' hd-tl-drop-3' by (smt)

next

fix ok_v and $inouts_v :: nat \Rightarrow real\ list$ and ok_v' and $inouts_v' :: nat \Rightarrow real\ list$ and $x :: nat$

assume a1: $\forall x. (x = 0 \longrightarrow$

$length(inouts_v\ 0) = 5 \wedge$

$length(inouts_v'\ 0) = 7 \wedge$

$inouts_v' x) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (take 3 (inouts_v x)), hd (tl (take 3 (inouts_v x))), hd (take 3 (inouts_v x)), 0, 0, 0,$
 $hd (tl (drop 3 (inouts_v (x - Suc 0))))] =$
 $inouts_v' x))$
from $a1$ **have** $len-5: \forall x. length(inouts_v x) = 5$
by (*metis neg0-conv*)
have $hd-take-3: hd (take 3 (inouts_v x)) = hd(inouts_v x)$
using $len-5$ **by** (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neg-numeral*)
have $hd-tl-take-3: hd (tl (take 3 (inouts_v x))) = hd (tl (inouts_v x))$
using $len-5$ **by** (*simp add: hd-tl-take-m*)
have $hd-drop-3: hd (drop 3 (inouts_v x)) = inouts_v x!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)
have $hd-drop-3': hd (drop 3 (inouts_v (x - Suc 0))) = inouts_v (x - Suc 0)!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)
have $hd-tl-drop-3: hd (tl (drop 3 (inouts_v x))) = inouts_v x!(4)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth nth-tl tl-drop*)
have $hd-tl-drop-3': hd (tl (drop 3 (inouts_v (x - Suc 0)))) = inouts_v (x - Suc 0)!(4)$
using $len-5$
by (*metis drop-Suc eval-nat-numeral(2) eval-nat-numeral(3) hd-drop-conv-nth lessI*
semiring-norm(26) semiring-norm(27) tl-drop)
show $(x = 0 \longrightarrow$
 $length(inouts_v 0) = 5 \wedge$
 $length(inouts_v' 0) = 7 \wedge$
 $[hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 0, 0, 0] = inouts_v' 0 \wedge$
 $(\neg inouts_v 0!(2) = 4 \longrightarrow$
 $length(inouts_v 0) = 5 \wedge$
 $length(inouts_v' 0) = 7 \wedge$
 $[hd (inouts_v 0), hd (tl (inouts_v 0)), hd (inouts_v 0), 0, 0, 0, 0] = inouts_v' 0) \wedge$
 $(0 < x \longrightarrow$
 $(inouts_v (x - Suc 0)!(3) = 0 \longrightarrow$
 $(inouts_v x!(2) = 8 \longrightarrow$
 $(inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1, 1, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1, 1, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v x!(2) = 8 \longrightarrow$
 $(inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0, 1, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0, 1, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x)) \wedge$

```

(¬ inoutsv (x - Suc 0)!(3) = 0 →
 inoutsv x!(2) = 8 →
 inoutsv (x - Suc 0)!(2) = 4 →
  length(inoutsv x) = 5 ∧
  length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 1, 1, 0, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧
 length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 0, 1, 0, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv x!(2) = 8 →
 inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧
 length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 1, 0, 0, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧
 length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 0, 0, 0, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x)))
using a1 hd-take-3 hd-tl-take-3 hd-drop-3' hd-tl-drop-3' by (smt )
next
fix okv and inoutsv::nat⇒real list and okv' and inoutsv'::nat⇒real list and x::nat
assume a1: ∀x. (x = 0 →
 length(inoutsv 0) = 5 ∧
 length(inoutsv' 0) = 7 ∧
 [hd (inoutsv 0), hd (tl (inoutsv 0)), hd (inoutsv 0), 0, 0, 0, 0] = inoutsv' 0 ∧
 (¬ inoutsv 0!(2) = 4 →
 length(inoutsv 0) = 5 ∧
 length(inoutsv' 0) = 7 ∧
 [hd (inoutsv 0), hd (tl (inoutsv 0)), hd (inoutsv 0), 0, 0, 0, 0] = inoutsv' 0)) ∧
(0 < x →
 (inoutsv (x - Suc 0)!(3) = 0 →
 inoutsv x!(2) = 8 →
 inoutsv (x - Suc 0)!(2) = 4 →
  length(inoutsv x) = 5 ∧
  length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 1, 1, 1, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧
 length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 0, 1, 1, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv x!(2) = 8 →
 inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧
 length(inoutsv' x) = 7 ∧
 [hd (inoutsv x), hd (tl (inoutsv x)), hd (inoutsv x), 1, 0, 1, inoutsv (x - Suc 0)!(4)] =
 inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
 length(inoutsv x) = 5 ∧

```

$length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0, 1, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x)) \wedge$
 $(\neg inouts_v (x - Suc 0)!(3) = 0 \longrightarrow$
 $(inouts_v x!(2) = 8 \longrightarrow$
 $(inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 1, 0, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 1, 0, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x)) \wedge$
 $(\neg inouts_v x!(2) = 8 \longrightarrow$
 $(inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 1, 0, 0, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x) \wedge$
 $(\neg inouts_v (x - Suc 0)!(2) = 4 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 7 \wedge$
 $[hd (inouts_v x), hd (tl (inouts_v x)), hd (inouts_v x), 0, 0, 0, inouts_v (x - Suc 0)!(4)] =$
 $inouts_v' x))))$
from $a1$ **have** $len-5: \forall x. length(inouts_v x) = 5$
by (*metis neq0-conv*)
have $hd-take-3: hd (take 3 (inouts_v x)) = hd(inouts_v x)$
using $len-5$ **by** (*metis append-take-drop-id hd-append2 take-eq-Nil zero-neq-numeral*)
have $hd-tl-take-3: hd (tl (take 3 (inouts_v x))) = hd (tl (inouts_v x))$
using $len-5$ **by** (*simp add: hd-tl-take-m*)
have $hd-drop-3: hd (drop 3 (inouts_v x)) = inouts_v x!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)
have $hd-drop-3': hd (drop 3 (inouts_v (x - Suc 0))) = inouts_v (x - Suc 0)!(3)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth*)
have $hd-tl-drop-3: hd (tl (drop 3 (inouts_v x))) = inouts_v x!(4)$
using $len-5$ **by** (*simp add: hd-drop-conv-nth nth-tl tl-drop*)
have $hd-tl-drop-3': hd (tl (drop 3 (inouts_v (x - Suc 0)))) = inouts_v (x - Suc 0)!(4)$
using $len-5$
by (*metis drop-Suc eval-nat-numeral(2) eval-nat-numeral(3) hd-drop-conv-nth lessI*
semiring-norm(26) semiring-norm(27) tl-drop)
show $(x = 0 \longrightarrow$
 $length(inouts_v 0) = 5 \wedge$
 $length(inouts_v' 0) = 7 \wedge$
 $[hd (take 3 (inouts_v 0)), hd (tl (take 3 (inouts_v 0))), hd (take 3 (inouts_v 0)), 0, 0, 0, 0] =$
 $inouts_v' 0 \wedge$
 $(\neg inouts_v 0!(2) = 4 \longrightarrow$
 $length(inouts_v 0) = 5 \wedge$
 $length(inouts_v' 0) = 7 \wedge$
 $[hd (take 3 (inouts_v 0)), hd (tl (take 3 (inouts_v 0))), hd (take 3 (inouts_v 0)), 0, 0, 0, 0] =$
 $inouts_v' 0)) \wedge$
 $(0 < x \longrightarrow$
 $(hd (drop 3 (inouts_v (x - Suc 0))) = 0 \longrightarrow$
 $(inouts_v x!(2) = 8 \longrightarrow$

```

(inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 1, 1, 1,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 0, 1, 1,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv x!(2) = 8 →
(inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 1, 0, 1,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 0, 0, 1,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ hd (drop 3 (inoutsv (x - Suc 0))) = 0 →
(inoutsv x!(2) = 8 →
(inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 1, 1, 0,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 0, 1, 0,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv x!(2) = 8 →
(inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 1, 0, 0,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x) ∧
(¬ inoutsv (x - Suc 0)!(2) = 4 →
length(inoutsv x) = 5 ∧
length(inoutsv' x) = 7 ∧
[hd (take 3 (inoutsv x)), hd (tl (take 3 (inoutsv x))), hd (take 3 (inoutsv x)), 0, 0, 0,
  hd (tl (drop 3 (inoutsv (x - Suc 0)))] =
inoutsv' x)))

```

using *a1 hd-take-3 hd-tl-take-3 hd-drop-3' hd-tl-drop-3'* by (*smt*)

qed

qed

then have *f6-2*: ... = ?*f6*
by (*smt Suc-eq-plus1 add-Suc-right numeral-Bit1 numeral-One one-add-one*)
have *simblock-f6*: *SimBlock 5 7 ?f6*
using *simblock-f3 simblock-f5 SimBlock-FBlock-parallel-comp*
by (*metis (no-types, lifting) Suc-1 Suc-eq-plus1 Suc-numeral add-numeral-left f6-0 f6-1 numeral-Bit1 numeral-One*)

have *ref-f6*: $((\forall n::nat \cdot (\langle\langle\lambda x n. ((hd(x n) = 0 \vee hd(x n) = 1))\rangle\rangle (\&inouts)_a (\langle n \rangle)_a)::sim\text{-state upred})$
 \vdash_n
 $((\forall n::nat \cdot ((\#_u(\$inouts (\langle n \rangle)_a)) =_u \langle 5 \rangle) \wedge ((\#_u(\$inouts' (\langle n \rangle)_a)) =_u \langle 7 \rangle) \wedge (head_u(\$inouts (\langle n \rangle)_a) =_u head_u(\$inouts' (\langle n \rangle)_a)) \wedge (head_u(tail_u(\$inouts (\langle n \rangle)_a)) =_u head_u(tail_u(\$inouts' (\langle n \rangle)_a))))))$
 $) \sqsubseteq$ *post-landing-finalize-part1*

proof –

have *1*: $((\forall n::nat \cdot (\langle\langle\lambda x n. ((hd(x n) = 0 \vee hd(x n) = 1))\rangle\rangle (\&inouts)_a (\langle n \rangle)_a)::sim\text{-state upred})$
 \vdash_n
 $((\forall n::nat \cdot ((\#_u(\$inouts (\langle n \rangle)_a)) =_u \langle 5 \rangle) \wedge ((\#_u(\$inouts' (\langle n \rangle)_a)) =_u \langle 7 \rangle) \wedge (head_u(\$inouts (\langle n \rangle)_a) =_u head_u(\$inouts' (\langle n \rangle)_a)) \wedge (head_u(tail_u(\$inouts (\langle n \rangle)_a)) =_u head_u(tail_u(\$inouts' (\langle n \rangle)_a))))))$
 $) \sqsubseteq$?*f6*

apply (*simp add: FBlock-def*)
apply (*rule ndesign-refine-intro*)
apply *simp*
apply (*rel-simp*)
apply (*rule conjI, clarify*)
apply (*metis gr-zeroI list.sel(1) list.sel(3)*)
apply (*clarify*)
by (*metis gr-zeroI list.sel(1) list.sel(3)*)

show ?*thesis*

using *1 f6 f6-0 f6-1 f6-2* **by** *simp*

qed

let ?*f7-f* = $(\lambda x n. [if\ hd(x\ n) = 0\ then\ 1\ else\ 0,\ hd(tl(x\ n))])$
let ?*f7* = *FBlock* $(\lambda x n. True) 2 2$?*f7-f*
have *f7*: $((LopNOT) \parallel_B (Id) (*\ door\ open\ time:\ double\ *)) = FBlock (\lambda x n. True) (1+1) (1+1)$
 $(\lambda x n. (((f-LopNOT \circ (\lambda xx nn. take\ 1\ (xx\ nn))) x\ n) \bullet ((f-Id \circ (\lambda xx nn. drop\ 1\ (xx\ nn)))) x\ n))$
using *SimBlock-LopNOT SimBlock-Id FBlock-parallel-comp*
by (*simp add: LopNOT-def simu-contract-real.Id-def*)
then have *f7-0*: ... = *FBlock* $(\lambda x n. True) 2 2$?*f7-f*

proof –

have $\forall x n. (\lambda x n. (((f-LopNOT \circ (\lambda xx nn. take\ 1\ (xx\ nn))) x\ n) \bullet ((f-Id \circ (\lambda xx nn. drop\ 1\ (xx\ nn)))) x\ n)) x\ n = ?f7-f\ x\ n$
by (*simp add: drop-Suc f-Id-def f-LopNOT-def hd-take-m*)

then show ?*thesis*

by (*simp add: numeral-2-eq-2*)

```

qed
have simblock-f7: SimBlock 2 2 (?f7)
  using SimBlock-LopNOT SimBlock-Id SimBlock-FBlock-parallel-comp
  by (metis (no-types, lifting) LopNOT-def f7 f7-0 one-add-one simu-contract-real.Id-def)

let ?f8-f = ( $\lambda x na.$  [if (if 1 ≤ (if hd(x na) = 0 then 1::real else 0) * 2
  then (if na = 0 then 0
  else min (vT-fd-sol-1
    ( $\lambda n1.$  ( $\lambda na.$  real-of-int
      (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
      ( $\lambda n1.$  (if hd(x n1) = 0 then 1::real else 0)) (na - 1))
      (( $\lambda na.$  real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
        (na - 1))) + 1
    else 0) > (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
  then 1 else 0])
let ?f8-f' = ( $\lambda x na.$  [if (if hd(x na) = 0
  then (if na = 0 then 0
  else min (vT-fd-sol-1
    ( $\lambda n1.$  ( $\lambda na.$  real-of-int
      (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
      ( $\lambda n1.$  (if hd(x n1) = 0 then 1::real else 0)) (na - 1))
      (( $\lambda na.$  real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0))
0])))
        (na - 1))) + 1
    else 0) > (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
  then 1 else 0])
let ?f8 = FBlock ( $\lambda x n.$  True) 2 1 ?f8-f'
have f8: ((LopNOT) ||B (Id) (* door-open-time: double *)) ;; variableTimer
  = ?f7 ;; variableTimer-simp-pat
  using variableTimer-simp f7 f7-0 by auto
then have f8-0: ... = FBlock ( $\lambda x n.$  True) 2 1 (variableTimer-simp-pat-f o ?f7-f)
  using simblock-f7 SimBlock-variableTimer-simp FBlock-seq-comp by blast
then have f8-1: ... = ?f8
proof -
  show ?thesis
  apply (simp add: FBlock-def)
  apply (rel-simp)
  apply (rule iffI)
  apply (clarify)
  defer
  apply (clarify)
  defer
  proof -
  fix okv and inoutsv::nat ⇒ real list and okv' and inoutsv'::nat ⇒ real list and x::nat
  assume a1:  $\forall x. (x = 0 \longrightarrow$ 
    (hd (inoutsv 0) = 0  $\longrightarrow$ 
      (int32 (RoundZero (real-of-int [Rate * max (hd (tl (inoutsv 0))) 0])) < 1  $\longrightarrow$ 
        length(inoutsv 0) = 2  $\wedge$  length(inoutsv' 0) = Suc 0  $\wedge$  [1] = inoutsv' 0)  $\wedge$ 
        ( $\neg$  int32 (RoundZero (real-of-int [Rate * max (hd (tl (inoutsv 0))) 0])) < 1  $\longrightarrow$ 
          length(inoutsv 0) = 2  $\wedge$  length(inoutsv' 0) = Suc 0  $\wedge$  [0] = inoutsv' 0)  $\wedge$ 
          ( $\neg$  hd (inoutsv 0) = 0  $\longrightarrow$ 
            (int32 (RoundZero (real-of-int [Rate * max (hd (tl (inoutsv 0))) 0])) < 0  $\longrightarrow$ 
              length(inoutsv 0) = 2  $\wedge$  length(inoutsv' 0) = Suc 0  $\wedge$  [1] = inoutsv' 0)  $\wedge$ 
              ( $\neg$  int32 (RoundZero (real-of-int [Rate * max (hd (tl (inoutsv 0))) 0])) < 0  $\longrightarrow$ 

```

$length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0))) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0\rceil)))$
 $< min\ (vT-fd-sol-1$
 $(\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ n1)))$
 $0\rceil))))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ (x - Suc\ 0)))$
 $0\rceil)))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0\rceil)))$
 $< min\ (vT-fd-sol-1$
 $(\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v$
 $n1)))\ 0\rceil))))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int$
 $(int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ (x - Suc\ 0)))$
 $0\rceil)))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x) \wedge$
 $(\neg\ hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0\rceil))) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0\rceil))) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x))$
from $a1$ **have** $len-2: \forall x.\ length(inouts_v\ x) = 2$
by $(metis\ (no-types,\ lifting)\ gr-zeroI)$
have $hd-tl-2: hd\ (tl\ (inouts_v\ x)) = inouts_v\ x!(Suc\ 0)$
using $len-2$
by $(metis\ Suc-1\ diff-Suc-1\ hd-conv-nth\ length-tl\ less-numeral-extra(1)\ list.size(3)$
 $nth-tl\ zero-neq-one)$
have $hd-tl-2': \forall x.\ hd\ (tl\ (inouts_v\ x)) = inouts_v\ x!(Suc\ 0)$
using $len-2$
by $(metis\ Suc-1\ diff-Suc-1\ hd-conv-nth\ length-tl\ less-numeral-extra(1)\ list.size(3)\ nth-tl$
 $zero-neq-one)$
have $hd-tl-2'': (hd\ (tl\ (inouts_v\ (x - Suc\ 0)))) = (inouts_v\ (x - Suc\ 0))!(Suc\ 0)$
using $len-2$ **using** $hd-tl-2'$ **by** $blast$
from $a1$ **have** $a1': \forall x.\ (x = 0 \longrightarrow$
 $(hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0\rceil))) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0\rceil))) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(\neg\ hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0\rceil))) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0\rceil))) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0))) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0\rceil)))$
 $< min\ (vT-fd-sol-1$
 $(\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ n1!(Suc\ 0))$
 $0\rceil))))$

$length(inouts_v x) = 2 \wedge length(inouts_v' x) = Suc\ 0 \wedge [0] = inouts_v' x))$
using $a1'$ **by** *blast*
next
fix ok_v **and** $inouts_v::nat \Rightarrow real\ list$ **and** ok_v' **and** $inouts_v':::nat \Rightarrow real\ list$ **and** $x::nat$
assume $a1: \forall x. (x = 0 \longrightarrow$
 $(hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0])) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0])) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(\neg\ hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0))) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1. real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v$
 $n1!(Suc\ 0))\ 0])))$
 $(\lambda n1. if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ (x - Suc\ 0))!(Suc\ 0)$
 $0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1. real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max$
 $(inouts_v\ n1!(Suc\ 0))\ 0])))$
 $(\lambda n1. if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ (x - Suc\ 0))!(Suc$
 $0))\ 0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x) \wedge$
 $(\neg\ hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x)))$
from $a1$ **have** $len-2: \forall x. length(inouts_v\ x) = 2$
by *(metis (no-types, lifting) gr-zeroI)*
have $hd-tl-2: hd\ (tl\ (inouts_v\ x)) = inouts_v\ x!(Suc\ 0)$
using $len-2$
by *(metis Suc-1 diff-Suc-1 hd-conv-nth length-tl less-numeral-extra(1) list.size(3)*
 $nth-tl zero-neq-one)$
have $hd-tl-2': \forall x. hd\ (tl\ (inouts_v\ x)) = inouts_v\ x!(Suc\ 0)$
using $len-2$
by *(metis Suc-1 diff-Suc-1 hd-conv-nth length-tl less-numeral-extra(1) list.size(3) nth-tl*
 $zero-neq-one)$
have $hd-tl-2'': (hd\ (tl\ (inouts_v\ (x - Suc\ 0)))) = (inouts_v\ (x - Suc\ 0))!(Suc\ 0)$
using $len-2$ **using** $hd-tl-2'$ **by** *blast*
from $a1$ **have** $a1': \forall x. (x = 0 \longrightarrow$
 $(hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0))))\ 0])) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg\ int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0))))\ 0])) < 1 \longrightarrow$

$length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(\neg hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl$
 $(inouts_v\ n1)))\ 0])))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ (x - Suc\ 0))))$
 $0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd$
 $(tl\ (inouts_v\ n1)))\ 0])))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ (x - Suc$
 $0))))\ 0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x)) \wedge$
 $(\neg hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [0] = inouts_v'\ x))$
using *hd-tl-2'* **by** *presburger*
show $(x = 0 \longrightarrow$
 $(hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 1 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(\neg hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [1] = inouts_v'\ 0) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ 0)))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ 0) = 2 \wedge length(inouts_v'\ 0) = Suc\ 0 \wedge [0] = inouts_v'\ 0)) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl$
 $(inouts_v\ n1)))\ 0])))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ (x - Suc\ 0))))$
 $0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 2 \wedge length(inouts_v'\ x) = Suc\ 0 \wedge [1] = inouts_v'\ x) \wedge$
 $(\neg real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl\ (inouts_v\ x)))\ 0])))$
 $< min\ (vT-fd-sol-1\ (\lambda n1.\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (hd\ (tl$
 $(inouts_v\ n1)))\ 0])))$
 $(\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0)\ (x - Suc\ 0))$

```

    (real-of-int (int32 (RoundZero (real-of-int [Rate * max (hd (tl (inouts_v (x - Suc
0)))) 0]))) +
      1 →
    length(inouts_v x) = 2 ∧ length(inouts_v' x) = Suc 0 ∧ [0] = inouts_v' x) ∧
    (¬ hd (inouts_v x) = 0 →
    (int32 (RoundZero (real-of-int [Rate * max (hd (tl (inouts_v x))) 0])) < 0 →
    length(inouts_v x) = 2 ∧ length(inouts_v' x) = Suc 0 ∧ [1] = inouts_v' x) ∧
    (¬ int32 (RoundZero (real-of-int [Rate * max (hd (tl (inouts_v x))) 0])) < 0 →
    length(inouts_v x) = 2 ∧ length(inouts_v' x) = Suc 0 ∧ [0] = inouts_v' x))
  using hd-tl-2' a1' by blast
qed
qed
then have f8-2: ... = FBlock (λx n. True) 2 1 ?f8-f'
proof -
  have ∀ x na. (1 ≤ (if hd(x na) = 0 then 1::real else 0) * 2) = (hd(x na) = 0)
  by simp
  then show ?thesis
  proof -
    have FBlock (λf n. True) 2 1 (λf n. [if
      real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (f n!(Suc 0)) 0])) <
      (if (1::real) ≤ (if hd (f n) = 0 then 1 else 0) * 2 then (if n = 0 then 0 else
      min (vT-fd-sol-1 (λn. real-of-int (int32 (RoundZero (real-of-int [(Rate::real) *
      max (f n!(Suc 0)) 0])))
      (λn. if hd (f n) = 0 then 1 else 0) (n - 1)) (real-of-int (int32 (RoundZero (real-of-int
      [(Rate::real) * max (f (n - 1)!(Suc 0)) 0]))) + 1 else 0) then 1 else 0]) =
      FBlock (λf n. True) 2 1 (λf n. [if real-of-int (int32 (RoundZero (real-of-int [(Rate::real)
      * max (f n!(Suc 0)) 0])) < (if hd (f n) = 0 then (if n = 0 then 0 else min (vT-fd-sol-1
      (λn. real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (f n!(Suc 0)) 0])))
      (λn. if hd (f n) = 0 then 1 else 0) (n - 1)) (real-of-int (int32 (RoundZero (real-of-int
      [(Rate::real) * max (f (n - 1)!(Suc 0)) 0]))) + 1 else 0) then 1 else 0]) ∨
      (∀ f n. [if real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (f n!(Suc 0)) 0])) <
      (if (1::real) ≤ (if hd (f n) = (0::real) then 1 else 0) * 2 then (if n = 0 then 0 else min
      (vT-fd-sol-1 (λn. real-of-int (int32 (RoundZero (real-of-int [(Rate::real) *
      max (f n!(Suc 0)) 0]))) (λn. if hd (f n) = 0 then 1 else 0) (n - 1))
      (real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (f (n - 1)!(Suc 0)) 0])))
      + 1 else 0)
      then 1::real else 0] = [if real-of-int (int32 (RoundZero (real-of-int [(Rate::real) *
      max (f n!(Suc 0)) 0])) < (if hd (f n) = 0 then (if n = 0 then 0 else min (vT-fd-sol-1
      (λn. real-of-int (int32 (RoundZero (real-of-int [(Rate::real) * max (f n!(Suc 0)) 0])))
      (λn. if hd (f n) = 0 then 1 else 0) (n - 1)) (real-of-int (int32 (RoundZero (real-of-int
      [(Rate::real) * max (f (n - 1)!(Suc 0)) 0]))) + 1 else 0) then 1 else 0])
    by auto
    then show ?thesis
    by force
  qed
qed
have simblock-f8: SimBlock 2 1 (FBlock (λx n. True) 2 1 ?f8-f')
  using simblock-f7 SimBlock-variableTimer-simp SimBlock-FBlock-seq-comp f8-0 f8-1 f8-2 by
fastforce

let ?f9-f = (λx n. [if (x n)!0 = 0 ∨ (x n)!1 = 0 ∨ (x n)!2 = 0 then 0 else 1,
  if (x n)!3 = 0 ∧ (x n)!4 = 0 then 0 else 1])
let ?f9 = FBlock (λx n. True) 5 2 ?f9-f
have f9: ((LopAND 3) ||B (LopOR 2)) = FBlock (λx n. True) (3+2) (1+1)
  (λx n. ((f-LopAND ∘ (λxx nn. take 3 (xx nn))) x n) •

```

```

      ((f-LopOR ◦ (λxx nn. drop 3 (xx nn)))) x n))
using SimBlock-LopAND SimBlock-LopOR FBlock-parallel-comp
by (simp add: LopAND-def LopOR-def)
then have f9-0: ... = FBlock (λx n. True) (3+2) (1+1) ?f9-f
proof –
  show ?thesis
    apply (simp add: FBlock-def f-LopAND-def f-LopOR-def)
    apply (rel-simp)
    apply (rule iffI)
    apply (clarify)
    defer
    apply (clarify)
    defer
    proof –
      fix okv and inoutsv::nat⇒real list and okv' and inoutsv'::nat⇒real list and x::nat
      assume a1: ∀x. (LOr (drop 3 (inoutsv x)) →
        (LAnd (take 3 (inoutsv x)) →
          length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [1, 1] = inoutsv' x) ∧
        (¬ LAnd (take 3 (inoutsv x)) →
          length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [0, 1] = inoutsv' x)) ∧
        (¬ LOr (drop 3 (inoutsv x)) →
          (LAnd (take 3 (inoutsv x)) →
            length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [1, 0] = inoutsv' x) ∧
            (¬ LAnd (take 3 (inoutsv x)) →
              length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [0, 0] = inoutsv' x))
          )
        )
      from a1 have len-5: ∀x. length(inoutsv x) = 5
      by blast
      have take-3: take 3 (inoutsv x) = [(inoutsv x)!0, (inoutsv x)!1, (inoutsv x)!2]
      using len-5 by (smt Cons-nth-drop-Suc Suc-1 Suc-eq-plus1 Suc-mono add-Suc-right
        add-diff-cancel-right' drop-0 numeral-3-eq-3 numeral-Bit1 numeral-eq-one-iff
        numeral-plus-one take-Suc-Cons take-eq-Nil zero-less-numeral)
      have land-take-3:
        LAnd (take 3 (inoutsv x)) = (¬ ((inoutsv x)!0 = 0 ∨ (inoutsv x)!1 = 0 ∨ (inoutsv x)!2
= 0))
      by (simp add: take-3)
      have drop-3: drop 3 (inoutsv x) = [(inoutsv x)!3, (inoutsv x)!4]
      using len-5
      by (metis Cons-nth-drop-Suc add-Suc cancel-ab-semigroup-add-class.add-diff-cancel-left'
        drop-eq-Nil eval-nat-numeral(2) eval-nat-numeral(3) lessI numeral-Bit0 order-refl pos2
        semiring-norm(26) semiring-norm(27) zero-less-diff)
      have lor-drop-3: LOr (drop 3 (inoutsv x)) = (¬((inoutsv x)!3 = 0 ∧ (inoutsv x)!4 = 0))
      by (simp add: drop-3)
      show (inoutsv x)!3 = 0 ∧ inoutsv x)!4 = 0 →
        (inoutsv x)!0 = 0 → length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [0, 0]
= inoutsv' x) ∧
        (inoutsv x)!0 = 0 → length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧
[0, 0] = inoutsv' x) ∧
        (inoutsv x)!2 = 0 → length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [0, 0]
= inoutsv' x) ∧
        (¬ inoutsv x)!0 = 0 ∧ ¬ inoutsv x)!0 = 0 ∧ ¬ inoutsv x)!2 = 0 →
        length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [1, 0] = inoutsv' x) ∧
        ((inoutsv x)!3 = 0 → ¬ inoutsv x)!4 = 0) →
        (inoutsv x)!0 = 0 → length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧ [0, 1]
= inoutsv' x) ∧
        (inoutsv x)!0 = 0 → length(inoutsv x) = 5 ∧ length(inoutsv' x) = Suc (Suc 0) ∧

```


$[0, 1] = \text{inouts}_v' x) \wedge$
 $(\text{inouts}_v x!(2) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 1]$
 $= \text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v x!(0) = 0 \wedge \neg \text{inouts}_v x!(\text{Suc } 0) = 0 \wedge \neg \text{inouts}_v x!(2) = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [1, 1] = \text{inouts}_v' x))$
using *land-take-3 lor-drop-3 a1 len-5 by simp*
next
fix ok_v **and** $\text{inouts}_v :: \text{nat} \Rightarrow \text{real list}$ **and** ok_v' **and** $\text{inouts}_v' :: \text{nat} \Rightarrow \text{real list}$ **and** $x :: \text{nat}$
assume $a1: \forall x. (\text{inouts}_v x!(3) = 0 \wedge \text{inouts}_v x!(4) = 0 \longrightarrow$
 $(\text{inouts}_v x!(0) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 0]$
 $= \text{inouts}_v' x) \wedge$
 $(\text{inouts}_v x!(\text{Suc } 0) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge$
 $[0, 0] = \text{inouts}_v' x) \wedge$
 $(\text{inouts}_v x!(2) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 0]$
 $= \text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v x!(0) = 0 \wedge \neg \text{inouts}_v x!(\text{Suc } 0) = 0 \wedge \neg \text{inouts}_v x!(2) = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [1, 0] = \text{inouts}_v' x)) \wedge$
 $((\text{inouts}_v x!(3) = 0 \longrightarrow \neg \text{inouts}_v x!(4) = 0) \longrightarrow$
 $(\text{inouts}_v x!(0) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 1]$
 $= \text{inouts}_v' x) \wedge$
 $(\text{inouts}_v x!(\text{Suc } 0) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge$
 $[0, 1] = \text{inouts}_v' x) \wedge$
 $(\text{inouts}_v x!(2) = 0 \longrightarrow \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 1]$
 $= \text{inouts}_v' x) \wedge$
 $(\neg \text{inouts}_v x!(0) = 0 \wedge \neg \text{inouts}_v x!(\text{Suc } 0) = 0 \wedge \neg \text{inouts}_v x!(2) = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [1, 1] = \text{inouts}_v' x))$
from $a1$ **have** $\text{len-5}: \forall x. \text{length}(\text{inouts}_v x) = 5$
by *blast*
have $\text{take-3}: \text{take } 3 (\text{inouts}_v x) = [(\text{inouts}_v x)!0, (\text{inouts}_v x)!1, (\text{inouts}_v x)!2]$
using len-5 **by** (*smt Cons-nth-drop-Suc Suc-1 Suc-eq-plus1 Suc-mono add-Suc-right*
add-diff-cancel-right' drop-0 numeral-3-eq-3 numeral-Bit1 numeral-eq-one-iff
numeral-plus-one take-Suc-Cons take-eq-Nil zero-less-numeral)
have $\text{land-take-3}:$
 $L\text{And} (\text{take } 3 (\text{inouts}_v x)) = (\neg ((\text{inouts}_v x)!0 = 0 \vee (\text{inouts}_v x)!1 = 0 \vee (\text{inouts}_v x)!2$
 $= 0))$
by (*simp add: take-3*)
have $\text{drop-3}: \text{drop } 3 (\text{inouts}_v x) = [(\text{inouts}_v x)!3, (\text{inouts}_v x)!4]$
using len-5
by (*metis Cons-nth-drop-Suc add-Suc cancel-ab-semigroup-add-class.add-diff-cancel-left'*
drop-eq-Nil eval-nat-numeral(2) eval-nat-numeral(3) lessI numeral-Bit0 order-refl pos2
semiring-norm(26) semiring-norm(27) zero-less-diff)
have $\text{lor-drop-3}: L\text{Or} (\text{drop } 3 (\text{inouts}_v x)) = (\neg ((\text{inouts}_v x)!3 = 0 \wedge (\text{inouts}_v x)!4 = 0))$
by (*simp add: drop-3*)
show $(L\text{Or} (\text{drop } 3 (\text{inouts}_v x)) \longrightarrow$
 $(L\text{And} (\text{take } 3 (\text{inouts}_v x)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\neg L\text{And} (\text{take } 3 (\text{inouts}_v x)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 1] = \text{inouts}_v' x)) \wedge$
 $(\neg L\text{Or} (\text{drop } 3 (\text{inouts}_v x)) \longrightarrow$
 $(L\text{And} (\text{take } 3 (\text{inouts}_v x)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [1, 0] = \text{inouts}_v' x) \wedge$
 $(\neg L\text{And} (\text{take } 3 (\text{inouts}_v x)) \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = \text{Suc}(\text{Suc } 0) \wedge [0, 0] = \text{inouts}_v' x))$
using *land-take-3 lor-drop-3 a1 len-5 by simp*
qed

```

qed
then have f9-1: ... = ?f9
  by (metis (no-types, lifting) Suc-eq-plus1 add-Suc nat-1-add-1 numeral-2-eq-2
    numeral-3-eq-3 numeral-code(3))
have simblock-f9: SimBlock 5 2 ?f9
  using SimBlock-LopAND SimBlock-LopOR SimBlock-FBlock-parallel-comp f9-0 f9-1 f9
  by (smt LopAND-def LopOR-def One-nat-def Suc-eq-plus1 add-Suc numeral-3-eq-3 numeral-Bit1
    one-add-one zero-less-numeral)

let ?f10-f = ( $\lambda x na. [latch-rec-calc-output$ 
  ( $\lambda n1. (if (x n1)!0 = 0 \vee (x n1)!1 = 0 \vee (x n1)!2 = 0$  then 0 else 1::real))
  ( $\lambda n1. (if (x n1)!3 = 0 \wedge (x n1)!4 = 0$  then 0 else 1::real))
  (na]])

let ?f10 = FBlock ( $\lambda x n. True$ ) 5 1 ?f10-f
have f10: (((LopAND 3)  $\parallel_B$  (LopOR 2)) ; ; latch) = ?f9 ; ; latch-simp-pat'
  using latch-simp f9 f9-0 f9-1 by simp
then have f10-0: ... = FBlock ( $\lambda x n. True$ ) 5 1 (latch-simp-pat-f' o ?f9-f)
  using simblock-f9 FBlock-seq-comp SimBlock-latch-simp' by blast
then have f10-1: ... = FBlock ( $\lambda x n. True$ ) 5 1 ?f10-f
proof –
  have 1:  $\forall x n. (latch-simp-pat-f' o ?f9-f) x n = ?f10-f x n$ 
    by (simp)
  then have 2: (latch-simp-pat-f' o ?f9-f) = ?f10-f
    using fun-eq by blast
  show ?thesis
    using 2 by (rule FBlock-eq)
qed
have simblock-f10: SimBlock 5 1 ?f10
  using simblock-f9 SimBlock-latch-simp' SimBlock-FBlock-seq-comp f10-0 f10-1 by fastforce

let ?f11-f = ( $\lambda x na. [if (if hd(x na) = 0$ 
  then (if na = 0 then 0
    else min (vT-fd-sol-1
      ( $\lambda n1. (\lambda na. real-of-int$ 
        (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
        ( $\lambda n1. (if hd(x n1) = 0$  then 1::real else 0)) (na - 1))
        (( $\lambda na. real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0))$ 
0])))
          (na - 1))) + 1
      else 0) > (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
    then 1 else 0,
    latch-rec-calc-output
    ( $\lambda n1. (if (x n1)!2 = 0 \vee (x n1)!3 = 0 \vee (x n1)!4 = 0$  then 0 else 1::real))
    ( $\lambda n1. (if (x n1)!5 = 0 \wedge (x n1)!6 = 0$  then 0 else 1::real))
    (na]])

let ?f11 = FBlock ( $\lambda x n. True$ ) 7 2 ?f11-f

have f11: ((((LopNOT)  $\parallel_B$  (Id) (* door-open-time: double *) ) ; ; variableTimer )
   $\parallel_B$ 
  (((LopAND 3)  $\parallel_B$  (LopOR 2)) ; ; latch))
  = ?f8  $\parallel_B$  ?f10
  using f10 f10-0 f10-1 f8 f8-0 f8-1 by auto
then have f11-0: ... = FBlock ( $\lambda x n. True$ ) (2+5) (1+1)
  ( $\lambda x n. (((?f8-f' o (\lambda xx nn. take 2 (xx nn))) x n) \bullet ((?f10-f o (\lambda xx nn. drop 2 (xx nn)))) x n)$ )
  using simblock-f8 simblock-f10 FBlock-parallel-comp by blast

```

```

then have f11-1: ... = FBlock (λx n. True) (2+5) (1+1) ?f11-f
proof -
  show ?thesis
  apply (rule FBlock-eq'')
  defer
  apply auto[1]
  apply auto[1]
  apply (rule allI)+
  apply (clarify)
proof -
  fix x::nat ⇒ real list and n::nat
  assume a1: ∀n. length(x n) = 2 + 5
  have hd-take-2: ∀n. hd (take 2 (x n)) = hd (x n)
    by (simp add: hd-take-m)
  have drop-2-0: ∀n. drop 2 (x n)!0 = (x n)!2
    using a1 by simp
  have drop-2-1: ∀n. drop 2 (x n)!1 = (x n)!3
    using a1 by simp
  have drop-2-1': ∀n. drop 2 (x n)!(Suc 0) = (x n)!3
    using a1 by simp
  have drop-2-2: ∀n. drop 2 (x n)!2 = (x n)!4
    using a1 by simp
  have drop-2-3: ∀n. drop 2 (x n)!3 = (x n)!5
    using a1 by simp
  have drop-2-4: ∀n. drop 2 (x n)!4 = (x n)!6
    using a1 by simp
  let ?lhs1 = ((λx na. [if real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na)!(Suc 0))
0])))
    < (if hd (x na) = 0
      then (if na = 0 then 0
        else min (vT-fd-sol-1
          (λn1. real-of-int
            (int32 (RoundZero (real-of-int [Rate * max (x n1)!(Suc 0)) 0])))
          (λn1. if hd (x n1) = 0 then 1 else 0) (na - 1))
        (real-of-int
          (int32 (RoundZero (real-of-int [Rate * max (x (na - 1))!(Suc 0))
0]))) +
          1
        else 0)
      then 1 else 0]) ∘ (λxx nn. take 2 (xx nn))) x n
  let ?rhs1 = (λx na. [if real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na)!(Suc 0))
0])))
    < (if hd (x na) = 0
      then (if na = 0 then 0
        else min (vT-fd-sol-1
          (λn1. real-of-int
            (int32 (RoundZero (real-of-int [Rate * max (x n1)!(Suc 0)) 0])))
          (λn1. if hd (x n1) = 0 then 1 else 0) (na - 1))
        (real-of-int
          (int32 (RoundZero (real-of-int [Rate * max (x (na - 1))!(Suc 0))
0]))) +
          1
        else 0)
      then 1 else 0]) x n
  let ?lhs2 = ((λx na. [latch-rec-calc-output

```

```

      (λn1. if x n1!(0) = 0 ∨ x n1!(1) = 0 ∨ x n1!(2) = 0 then 0 else 1::real)
      (λn1. if x n1!(3) = 0 ∧ x n1!(4) = 0 then 0 else 1::real) (na)]
    ◦ (λxx nn. drop 2 (xx nn)) x n
  let ?rhs2 = (λx n. [latch-rec-calc-output
    (λn1. if x n1!(2) = 0 ∨ x n1!(3) = 0 ∨ x n1!(4) = 0 then 0 else 1::real)
    (λn1. if x n1!(5) = 0 ∧ x n1!(6) = 0 then 0 else 1::real) (n)]) x n
  let ?rhs1' = if real-of-int (int32 (RoundZero (real-of-int [Rate * max (x n!(Suc 0)) 0])))
    < (if hd (x n) = 0
      then (if n = 0 then 0
        else min (vT-fd-sol-1
          (λn1. real-of-int
            (int32 (RoundZero (real-of-int [Rate * max (x n1!(Suc 0)) 0])))
            (λn1. if hd (x n1) = 0 then 1 else 0) (n - 1))
          (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x (n - 1)!(Suc 0))
0]))))) +
      1::real
    else 0)
  then 1::real else 0
  let ?rhs2' = latch-rec-calc-output
    (λn1. if x n1!(2) = 0 ∨ x n1!(3) = 0 ∨ x n1!(4) = 0 then 0 else 1::real)
    (λn1. if x n1!(5) = 0 ∧ x n1!(6) = 0 then 0 else 1::real) (n)
  from a1 hd-take-2 have f1: ?lhs1 = ?rhs1
  by (simp)
  have 11: ∀ na. (λn1. if drop 2 (x n1)!(0) = 0 ∨ drop 2 (x n1)!(Suc 0) = 0 ∨ drop 2 (x
n1)!(2) = 0 then 0 else 1) na
    = (λn1. if x n1!(2) = 0 ∨ x n1!(3) = 0 ∨ x n1!(4) = 0 then 0 else 1) na
  using drop-2-0 drop-2-1' drop-2-2 drop-2-3 drop-2-4 a1 by simp
  then have 12: (λn1. if drop 2 (x n1)!(0) = 0 ∨ drop 2 (x n1)!(Suc 0) = 0 ∨ drop 2 (x
n1)!(2) = 0 then 0 else 1)
    = (λn1. if x n1!(2) = 0 ∨ x n1!(3) = 0 ∨ x n1!(4) = 0 then 0 else 1)
  by (rule fun-eq)
  have 21: ∀ na. (λn1. if drop 2 (x n1)!(3) = 0 ∧ drop 2 (x n1)!(4) = 0 then 0 else 1) na
    = (λn1. if x n1!(5) = 0 ∧ x n1!(6) = 0 then 0 else 1) na
  using drop-2-0 drop-2-1' drop-2-2 drop-2-3 drop-2-4 a1 by simp
  then have 22: (λn1. if drop 2 (x n1)!(3) = 0 ∧ drop 2 (x n1)!(4) = 0 then 0 else 1)
    = (λn1. if x n1!(5) = 0 ∧ x n1!(6) = 0 then 0 else 1)
  by (rule fun-eq)
  have latch-eq:
    latch-rec-calc-output (λn1. if drop 2 (x n1)!(0) = 0 ∨ drop 2 (x n1)!(Suc 0) = 0
      ∨ drop 2 (x n1)!(2) = 0 then 0 else 1)
      (λn1. if drop 2 (x n1)!(3) = 0 ∧ drop 2 (x n1)!(4) = 0 then 0 else 1) (n - Suc 0)
    = latch-rec-calc-output (λn1. if x n1!(2) = 0 ∨ x n1!(3) = 0 ∨ x n1!(4) = 0 then 0 else 1)
      (λn1. if x n1!(5) = 0 ∧ x n1!(6) = 0 then 0 else 1) (n - Suc 0)
  by (simp add: 12 22)
  have f2: ?lhs2 = ?rhs2
  apply (simp)
  using latch-eq drop-2-0 drop-2-1 drop-2-2 drop-2-3 drop-2-4 a1
  using numeral-1-eq-Suc-0 numerals(1) by presburger
  have f12: (?lhs1 • ?lhs2) = ?rhs1 • ?rhs2
  using f1 f2 by simp
  then have f21: ... = [?rhs1', ?rhs2']
  by simp
  show (?lhs1 • ?lhs2) = [?rhs1', ?rhs2']
  using f12 f21 by (simp)
qed

```

```

qed
then have f11-2: ... = ?f11
  by (smt Suc-eq-plus1 add-Suc-right numeral-Bit1 numeral-One one-add-one)
have simblock-f11: SimBlock 7 2 ?f11
  using simblock-f8 simblock-f10 SimBlock-FBlock-parallel-comp
  by (smt Suc-numeral add.commute add-Suc-right add-numeral-left f11-0 f11-1 numeral-Bit1
    numeral-One one-add-one)

let ?f12-f-1 =  $\lambda x na.$  if (if hd(x na) = 0
  then (if na = 0 then 0
    else min (vT-fd-sol-1
      ( $\lambda n1.$  ( $\lambda na.$  real-of-int
        (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0]))) n1)
      ( $\lambda n1.$  (if hd(x n1) = 0 then 1::real else 0)) (na - 1))
      (( $\lambda na.$  real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0))
0])))
        (na - 1))) + 1::real
    else 0) > (real-of-int (int32 (RoundZero (real-of-int [Rate * max (x na!(Suc 0)) 0])))
  then 1::real else 0)
let ?f12-f-2 =  $\lambda x na.$  latch-rec-calc-output
  ( $\lambda n1.$  (if hd(x n1) = 0  $\vee$  (if (n1 > 0  $\wedge$  (x (n1-1))!2 = 4) then 1::real else 0) = 0
 $\vee$  (if (x n1)!2 = 8 then 1::real else 0) = 0 then 0 else 1::real))
  ( $\lambda n1.$  (if ((if n1 = 0 then 0 else (if (x (n1 - 1))!3 = 0 then 1::real else 0))) = 0  $\wedge$ 
    (if n1 = 0 then 0 else (x (n1 - 1))!4) = 0 then 0 else 1::real))
  (na))
let ?f12-f-2' =  $\lambda x na.$  (latch-rec-calc-output
  ( $\lambda n1.$  (if hd(x n1) = 0  $\vee$  n1 = 0  $\vee$  (x (n1-1))!2  $\neq$  4  $\vee$  (x n1)!2  $\neq$  8
    then 0 else 1::real))
  ( $\lambda n1.$  (if ((n1 = 0)  $\vee$  ((x (n1 - 1))!3  $\neq$  0  $\wedge$  (x (n1 - 1))!4 = 0))
    then 0 else 1::real))
  (na))
let ?f12-f = ( $\lambda x na.$  [?f12-f-1 x na, ?f12-f-2 x na])
let ?f12 = FBlock ( $\lambda x n.$  True) 5 2 ?f12-f
let ?f12-f' = ( $\lambda x na.$  [?f12-f-1 x na, ?f12-f-2' x na])
let ?f12' = FBlock ( $\lambda x n.$  True) 5 2 ?f12-f'
have f12-f-2-eq:  $\forall x n.$  ?f12-f-2 x n = ?f12-f-2' x n
  apply (rule allI)+
  apply (simp)
  apply (induct-tac n)
  apply auto[1]
  by simp
have f12: (
  (
    (
      (
        Split2 (* door-closed (boolean, 1/10s) is split into two *)
        ||B
        Id (* door-open-time: double *)
      ) ; ; Router 3 [0,2,1]
    )
    ||B
    post-mode
  )
  ||B
  (

```

```

    (UnitDelay 1.0 ;; LopNOT)
  ||B
  (UnitDelay 0) (* Delay2 *)
)
);;
(
  (
    (LopNOT)
  ||B
  (Id) (* door-open-time: double *)
  ) ;; variableTimer
)
||B
(
  (
    (LopAND 3)
  ||B
  (LopOR 2)
  ) ;; latch
)
) = ?f6 ;; ?f11
using f11 f11-0 f11-1 f11-2 f8 f8-0 f8-1 f6 f6-0 f6-1 f6-2 by auto
then have f12-0: ... = FBlock (λx n. True) 5 2 (?f11-f o ?f6-f)
using simblock-f6 simblock-f11 FBlock-seq-comp by blast
then have f12-1: ... = FBlock (λx n. True) 5 2 (?f12-f)
proof -
  have hd-tl-eq: ∀ x n. length(x n) > 1 → hd (tl (x n)) = (x n)!(Suc 0)
  by (metis One-nat-def drop-0 drop-Suc hd-drop-conv-nth)
show ?thesis
  apply (rule FBlock-eq'')
  defer
  apply auto[1]
  apply auto[1]
  apply (simp)
  apply (rule allI)+
  apply (clarify)
  apply (rule conjI)
  apply (simp add: hd-tl-eq)
  apply (clarify, rule conjI)
  defer
  apply (simp add: hd-tl-eq)
  proof -
    fix x::nat ⇒ real list and n::nat
    assume a1: ∀ na. length(x na) = 5
    have vT-eq: (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl
(x n1))) 0⌋))))
      (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
      = (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x n1)!(Suc 0)
0⌋))))
      (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
    by (simp add: hd-tl-eq a1)
    have real-eq: real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl (x n))) 0⌋)))
      = real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x n)!(Suc 0) 0⌋)))
    by (simp add: hd-tl-eq a1)

```

```

show a2: hd (x n) = 0 →
  (real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl (x n))) 0⌋))))
  < min (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl
(x n1))) 0⌋))))
    (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
    (real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl (x (n - Suc 0)))) 0⌋))))
+
  1 →
  real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x n!(Suc 0)) 0⌋)))
  < min (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x n1!(Suc
0)) 0⌋))))
    (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
    (real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x (n - Suc 0)!(Suc 0)) 0⌋))))
+
  1) ∧
  (¬ real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl (x n))) 0⌋)))
  < min (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl
(x n1))) 0⌋))))
    (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
    (real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (hd (tl (x (n - Suc 0))))
0⌋)))) +
  1 →
  ¬ real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x n!(Suc 0)) 0⌋)))
  < min (vT-fd-sol-1 (λn1. real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x
n1!(Suc 0)) 0⌋))))
    (λn1. if hd (x n1) = 0 then 1 else 0) (n - Suc 0))
    (real-of-int (int32 (RoundZero (real-of-int ⌈Rate * max (x (n - Suc 0)!(Suc 0))
0⌋)))) +
  1)
using vT-eq real-eq a1 hd-tl-eq
by (simp add: hd-tl-eq)
qed
qed
then have f12-2: ... = FBlock (λx n. True) 5 2 (?f12-f')
proof -
  show ?thesis
  apply (rule FBlock-eq'')
  using f12-f-2-eq apply blast
  apply simp
  by simp
qed
have simblock-f12: SimBlock 5 2 ?f12'
  using simblock-f6 simblock-f11 FBlock-seq-comp SimBlock-FBlock-seq-comp f12-0 f12-1 f12-2
  by smt

let ?f13-f = (λx n. [if ((hd(x n) ≠ 0 ∧ hd(tl(x n)) ≠ 0) ∧
  (n > 0 ∧ (hd(x (n-1)) = 0 ∨ hd(tl(x (n-1))) = 0))) then 1 else 0])
let ?f13 = FBlock (λx n. True) 2 1 ?f13-f
have f13: LopAND 2;; rise1Shot = LopAND 2;; rise1Shot-simp-pat
  by (simp add: rise1Shot-simp)
then have f13-0: ... = FBlock (λx n. True) 2 1 (rise1Shot-simp-pat-f o f-LopAND)
  using SimBlock-rise1Shot-simp SimBlock-LopAND FBlock-seq-comp
  by (simp add: LopAND-def)
then have f13-1: ... = ?f13
proof -

```

```

show ?thesis
  apply (rule FBlock-eq'')
  defer
  apply (simp add: f-LopAND-def)
  apply (simp add: f-LopAND-def)
  apply (rule allI)+
  apply (clarify)
  apply (simp add: f-LopAND-def)
  apply (clarify)
  proof -
    fix x:: nat  $\Rightarrow$  real list and n::nat
    assume a1:  $\forall n. \text{length}(x\ n) = 2$ 
    assume a2:  $n > 0$ 
    from a1 a2 have land-1: LAnd (x (n - Suc 0)) =
      ( $\neg \text{hd}(x\ (n - \text{Suc}\ 0)) = 0 \wedge \neg \text{hd}(\text{tl}(x\ (n - \text{Suc}\ 0))) = 0$ )
    using LAnd.simps(1) LAnd.simps(2) append-eq-Cons-conv hd-Cons-tl length-Cons list.sel(3)

      list-equal-size2 tl-append2 by smt
    from a1 a2 have land-2: LAnd (x n) =
      ( $\neg \text{hd}(x\ n) = 0 \wedge \neg \text{hd}(\text{tl}(x\ n)) = 0$ )
    using LAnd.simps(1) LAnd.simps(2) append-eq-Cons-conv hd-Cons-tl length-Cons list.sel(3)

      list-equal-size2 tl-append2 by smt
    show (LAnd (x (n - Suc 0))  $\longrightarrow$ 
       $\text{hd}(x\ n) = 0 \vee \text{hd}(\text{tl}(x\ n)) = 0 \vee \neg \text{hd}(x\ (n - \text{Suc}\ 0)) = 0 \wedge \neg \text{hd}(\text{tl}(x\ (n - \text{Suc}\ 0)))$ 
      = 0)  $\wedge$ 
      ( $\neg$  LAnd (x (n - Suc 0))  $\longrightarrow$ 
      (LAnd (x n)  $\longrightarrow$ 
       $\neg \text{hd}(x\ n) = 0 \wedge \neg \text{hd}(\text{tl}(x\ n)) = 0 \wedge (\text{hd}(x\ (n - \text{Suc}\ 0)) = 0 \vee \text{hd}(\text{tl}(x\ (n - \text{Suc}\ 0))) = 0$ ))  $\wedge$ 
      ( $\neg$  LAnd (x n)  $\longrightarrow$ 
       $\text{hd}(x\ n) = 0 \vee \text{hd}(\text{tl}(x\ n)) = 0 \vee \neg \text{hd}(x\ (n - \text{Suc}\ 0)) = 0 \wedge \neg \text{hd}(\text{tl}(x\ (n - \text{Suc}\ 0))) = 0$ )) = 0))
    using land-1 land-2 by blast
  qed
qed
have simblock-f13: SimBlock 2 1 ?f13
  using SimBlock-rise1Shot-simp SimBlock-LopAND SimBlock-FBlock-seq-comp
  by (metis (no-types, lifting) LopAND-def f13-0 f13-1 pos2)

let ?f14-f = ( $\lambda x\ n. [\text{if } ((\text{hd}(x\ n) \neq 0 \wedge \text{hd}(\text{tl}(x\ n)) \neq 0) \wedge$ 
  ( $n > 0 \wedge (\text{hd}(x\ (n-1)) = 0 \vee \text{hd}(\text{tl}(x\ (n-1))) = 0$ )) then 1 else 0,
   $\text{if } ((\text{hd}(x\ n) \neq 0 \wedge \text{hd}(\text{tl}(x\ n)) \neq 0) \wedge$ 
  ( $n > 0 \wedge (\text{hd}(x\ (n-1)) = 0 \vee \text{hd}(\text{tl}(x\ (n-1))) = 0$ )) then 1 else 0])
let ?f14 = FBlock ( $\lambda x\ n. \text{True}$ ) 2 2 ?f14-f
have f14: LopAND 2;; rise1Shot ;; Split2 = ?f13 ;; Split2
  by (metis RA1 f13-0 f13-1 rise1Shot-simp)
then have f14-0: ... = FBlock ( $\lambda x\ n. \text{True}$ ) 2 2 (f-Split2 o ?f13-f)
  using simblock-f13 SimBlock-Split2 FBlock-seq-comp
  by (simp add: Split2-def)
then have f14-1: ... = ?f14
  proof -
    show ?thesis
    apply (rule FBlock-eq)
    using f-Split2-def

```



```

    by fastforce
  qed
  have simblock-f14: SimBlock 2 2 ?f14
    using simblock-f13 SimBlock-Split2 SimBlock-FBlock-seq-comp
    by (metis (no-types, lifting) Split2-def f14-0 f14-1)

  let ?f15-f = ( $\lambda x n.$  [if (((?f12-f-1 x n)  $\neq$  0  $\wedge$  (?f12-f-2' x n)  $\neq$  0)  $\wedge$ 
    (n > 0  $\wedge$  ((?f12-f-1 x (n-1)) = 0  $\vee$  (?f12-f-2' x (n-1)) = 0)) then 1 else 0,
    if (((?f12-f-1 x n)  $\neq$  0  $\wedge$  (?f12-f-2' x n)  $\neq$  0)  $\wedge$ 
    (n > 0  $\wedge$  ((?f12-f-1 x (n-1)) = 0  $\vee$  (?f12-f-2' x (n-1)) = 0)) then 1 else 0])
  let ?f15 = FBlock ( $\lambda x n.$  True) 5 2 ?f15-f
  have f15: (
    (
      (
        (
          Split2 (* door-closed (boolean, 1/10s) is split into two *)
           $\parallel_B$ 
          Id (* door-open-time: double *)
        ) ;; Router 3 [0,2,1]
      )
       $\parallel_B$ 
      post-mode
    )
     $\parallel_B$ 
    (
      (UnitDelay 1.0 ;; LopNOT)
       $\parallel_B$ 
      (UnitDelay 0) (* Delay2 *)
    )
  ) ;;
  (
    (
      (LopNOT)
       $\parallel_B$ 
      (Id) (* door-open-time: double *)
    ) ;; variableTimer
  )
   $\parallel_B$ 
  (
    (
      (LopAND 3)
       $\parallel_B$ 
      (LopOR 2)
    ) ;; latch
  )
  ) ;; LopAND 2 ;; rise1Shot ;; Split2) = ?f12' ;; ?f14
  by (smt RA1 f12 f12-0 f12-1 f12-2 f14 f14-0 f14-1)
  then have f15-0: ... = FBlock ( $\lambda x n.$  True) 5 2 (?f14-f o ?f12-f')
  using simblock-f14 simblock-f12 FBlock-seq-comp by blast
  then have f15-1: ... = ?f15
  proof -
    have 1:  $\forall x n.$  ((?f14-f o ?f12-f') x n = ?f15-f x n)
    apply (rule allI)+

```

```

    by (simp)
  have 2: (?f14-f o ?f12-f') = ?f15-f
    using 1 fun-eq by blast
  show ?thesis
    apply (rule FBlock-eq)
    using 1 2 by blast
qed
have simblock-f15: SimBlock 5 2 ?f15
  using simblock-f14 simblock-f12 SimBlock-FBlock-seq-comp f15-0 f15-1
  by (metis (no-types, lifting))
have inps-f15: inps ?f15 = 5
  using simblock-f15 inps-P by blast
have outps-f15: outps ?f15 = 2
  using simblock-f15 outps-P by blast

have f16: post-landing-finalize-1 = ?f15 fD (4, 1)
  using f15 f15-0 f15-1 post-landing-finalize-1-def by presburger
show ?thesis
  apply (simp only: plf-rise1shot-simp-def)
  using f16 simblock-f15 by presburger
qed

```

Finally, *post-landing-finalize-1* is simplified to a design with a feedback.

```

lemma post-landing-finalize-1-simp:
  post-landing-finalize-1 = plf-rise1shot-simp fD (4, 1)
  using post-landing-finalize-1-simp-simblock by blast

```

```

lemma post-landing-finalize-1-simblock:
  SimBlock 5 2 plf-rise1shot-simp
  using post-landing-finalize-1-simp-simblock by blast

```

```

lemma inps-plf-rise1shot:
  inps plf-rise1shot-simp = 5
  using post-landing-finalize-1-simblock inps-P by blast

```

```

lemma outps-plf-rise1shot:
  outps plf-rise1shot-simp = 2
  using post-landing-finalize-1-simblock outps-P by blast

```

C.5 Verification

Here we assume the maximum door open time is 1000s. It could be a value less than 214748364.

abbreviation *max-door-open-time* \equiv 1000

C.5.1 Requirement 01

post-landing-finalize-req-01: A finalize event will be broadcast after the aircraft door has been open continuously for *door-open-time* seconds while the aircraft is on the ground after a successful landing.

Here we assume the constant door open time is 20s. It should be a variable but according to Assumption 3, it does not change while the aircraft is on the ground. So we can regard it as a constant after landing.

abbreviation *c-door-open-time* \equiv 20

req-01-contract is the requirement to be verified. Its precondition specifies that *door-closed* and *ac-on-ground* are boolean and *door-open-time* is constant. Its postcondition specifies that

- it always has four inputs and one output;
- the requirement:
 - after a successful landing: door is closed, aircraft is on ground, mode is switched from LANDING (at step m) to GROUND (at step $m + 1$);
 - then the door has been open continuously for *door-open-time* (200): from step $m+2+p$ to $m + 2 + p + \text{door_open_time}$ ($m + 2 + p + 200$), therefore the door is closed at the step before p ;
 - while the aircraft is on ground: *ac-on-ground* is true and *mode=GROUND*;
 - additionally, between step m and p , the *finalize-event* is not enabled;
 - then a *finalize-event* will be broadcast at step $p + \text{door_open_time}$

definition *req-01-contract* $\equiv ((\forall n::\text{nat} \cdot ($
 $\ll(\lambda x n.$
 $($
 $(hd(x n) = 0 \vee hd(x n) = 1) \wedge (* \text{ door-closed is boolean } *)$
 $((x n)!1 = c\text{-door-open-time}) \wedge (* \text{ door-open-time } *)$
 $((x n)!3 = 0 \vee (x n)!3 = 1) (* \text{ ac-on-ground is boolean } *)$
 $)) \gg (\&\text{inouts})_a (\ll n \gg)_a :: \text{sim-state upred}$
 \vdash_n
 $((\forall n::\text{nat} \cdot$
 $((\#_u(\$inouts (\ll n \gg)_a)) =_u \ll 4 \gg) \wedge$
 $((\#_u(\$inouts' (\ll n \gg)_a)) =_u \ll 1 \gg)) \wedge$
 $(* m : \text{LANDING}$
 $m+1 : \text{GROUND}$
 $\dots : \neg \text{finalize-event during this time, door may be open for a while but not longer like}$
 door-open-time
 $p-1 : \text{door closed}$
 $p[0] : \text{door open}$
 $\dots : \text{door continuously open}$
 $p[n] : \text{door open for door-open-time seconds, finalize-event enabled.}$
 $*)$
 $(\forall m::\text{nat} \cdot$
 $($
 $((* \text{ A successful landing } *)$
 $((\ll nth \gg (\$inouts (\ll m \gg)_a)_a (\mathcal{I})_a =_u 1) (* \text{ ac-on-ground = true*})$
 $\wedge (\ll nth \gg (\$inouts (\ll m \gg)_a)_a (\mathcal{I})_a =_u 4) (* \text{ mode = LANDING } *)$
 $\wedge (\ll nth \gg (\$inouts (\ll m \gg)_a)_a (\mathcal{I})_a =_u 1) (* \text{ door-closed = true } *)$
 $) \wedge$
 $((\ll nth \gg (\$inouts (\ll m+1 \gg)_a)_a (\mathcal{I})_a =_u 1) (* \text{ ac-on-ground = true*})$
 $\wedge (\ll nth \gg (\$inouts (\ll m+1 \gg)_a)_a (\mathcal{I})_a =_u 8) (* \text{ mode = GROUND } *)$
 $\wedge (\ll nth \gg (\$inouts (\ll m+1 \gg)_a)_a (\mathcal{I})_a =_u 1) (* \text{ door-closed = true } *)$
 $)$
 $) \Rightarrow$
 $((* \text{ The door is open continuously for door-open-time seconds from } (m+p) *)$
 $\forall p::\text{nat} \cdot$
 $($
 $(\forall q::\text{nat} \cdot$
 $((\ll q \gg \leq_u \ll c\text{-door-open-time} * \text{Rate} \gg)) \Rightarrow$

$$\begin{aligned}
& (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+2+p+q \rangle\rangle)_a)_a (0)_a =_u 0) (* door-closed = false *) \\
&) (* The door is continuously open *) \\
&) \wedge \\
& (\forall q::nat \cdot (\langle\langle q \rangle\rangle \leq_u \langle\langle p + c-door-open-time \rangle\rangle * Rate) \Rightarrow \\
& \quad (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+2+q \rangle\rangle)_a)_a (3)_a =_u 1) (* ac-on-ground = true *) \wedge \\
& \quad (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+2+q \rangle\rangle)_a)_a (2)_a =_u 8) (* mode = GROUND *)) \\
&) (* the aircraft is always on the ground from m+2 to m+p+times *) \wedge \\
& (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+2+p-1 \rangle\rangle)_a)_a (0)_a =_u 1) (* door-closed = true before \$p\$ *) \wedge \\
& (\forall q::nat \cdot (\langle\langle q \rangle\rangle <_u \langle\langle p \rangle\rangle) \Rightarrow (head_u((\$inouts' (\langle\langle m+2+q \rangle\rangle)_a)) =_u 0))) \\
& \quad (* finalize-event has not been enabled before p *) \\
& \Rightarrow (\$inouts' (\langle\langle m + 2 + p + c-door-open-time \rangle\rangle)_a) =_u \langle 1 \rangle (* then the finalize-event \\
is true. *) \\
&) \\
&) \\
&))))
\end{aligned}$$

req-01-1-contract is the contract for *post-landing-finalize-1* without feedback: *plf-rise1shot-simp*. It is similar to *req-01-contract* except that 1) it has five inputs and two outputs (the feedback operator will remove one input and one output); 2) the 2nd output is equal to the 4th input since they are connected together by the feedback loop.

definition *req-01-1-contract* \equiv $(\forall n::nat \cdot ($

$$\begin{aligned}
& \langle\langle \lambda x n. \\
& \quad (\\
& \quad \quad (hd(x n) = 0 \vee hd(x n) = 1) \wedge (* door-closed is boolean *) \\
& \quad \quad ((x n)!1 = c-door-open-time) \wedge (* door-open-time *) \\
& \quad \quad ((x n)!3 = 0 \vee (x n)!3 = 1) (* ac-on-ground is boolean *) \\
& \quad \quad) \rangle\rangle (\&inouts)_a (\langle\langle n \rangle\rangle)_a :: sim-state upred) \\
& \vdash_n \\
& (\forall n::nat \cdot \\
& \quad (\#_u(\$inouts (\langle\langle n \rangle\rangle)_a)) =_u \langle\langle 5 \rangle\rangle) \wedge \\
& \quad (\#_u(\$inouts' (\langle\langle n \rangle\rangle)_a)) =_u \langle\langle 2 \rangle\rangle) \wedge \\
& \quad (* m : LANDING \\
& \quad \quad m+1 : GROUND \\
& \quad \quad \dots : \neg finalize-event \text{ during this time, door may be open for a while but not longer like} \\
& \quad \quad \quad \text{door-open-time} \\
& \quad \quad p-1 : door closed \\
& \quad \quad p[0] : door open \\
& \quad \quad \dots : door continuously open \\
& \quad \quad p[n] : door open for door-open-time seconds, finalize-event enabled. \\
& *) \\
& (\forall m::nat \cdot \\
& \quad (\\
& \quad \quad (* A successful landing *) \\
& \quad \quad (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m \rangle\rangle)_a)_a (3)_a =_u 1) (* ac-on-ground = true *) \\
& \quad \quad \wedge (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m \rangle\rangle)_a)_a (2)_a =_u 4) (* mode = LANDING *) \\
& \quad \quad \wedge (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m \rangle\rangle)_a)_a (0)_a =_u 1) (* door-closed = true *) \\
& \quad \quad) \wedge \\
& \quad \quad (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+1 \rangle\rangle)_a)_a (3)_a =_u 1) (* ac-on-ground = true *) \\
& \quad \quad \wedge (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+1 \rangle\rangle)_a)_a (2)_a =_u 8) (* mode = GROUND *) \\
& \quad \quad \wedge (\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+1 \rangle\rangle)_a)_a (0)_a =_u 1) (* door-closed = true *) \\
& \quad \quad) \wedge \\
& \quad \quad (\forall n::nat \cdot (head_u(tail_u(\$inouts' (\langle\langle n \rangle\rangle)_a)) =_u \langle\langle nth \rangle\rangle (\$inouts (\langle\langle n \rangle\rangle)_a)_a (4)_a)) \\
& \quad \quad (* 4th input is equal to output *) \\
& \quad \quad) \Rightarrow \\
& \quad \quad (* The door is open continuously for door-open-time seconds from (m+p) *) \\
\end{aligned}$$

$(x = 0 \longrightarrow \text{length}(\text{inouts}_v\ 0) = 5 \wedge \text{length}(\text{inouts}_v'\ 0) = 2 \wedge [0, 0] = \text{inouts}_v'\ 0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd}(\text{inouts}_v\ x) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int}(\lceil \text{Rate} * \max(\text{inouts}_v\ x!(\text{Suc}\ 0))\ 0\rceil))))$
 $< \min\ 1(\text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}(\lceil \text{Rate} * \max(\text{inouts}_v\ 0!(\text{Suc}\ 0))\ 0\rceil)))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v\ (n1 - \text{Suc}\ 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v\ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v\ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v\ (n1 - \text{Suc}\ 0)!3 = 0 \wedge \text{inouts}_v\ (n1 - \text{Suc}\ 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v\ x) = 5 \wedge \text{length}(\text{inouts}_v'\ x) = 2 \wedge [1, 1] = \text{inouts}_v'\ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v\ (n1 - \text{Suc}\ 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v\ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v\ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v\ (n1 - \text{Suc}\ 0)!3 = 0 \wedge \text{inouts}_v\ (n1 - \text{Suc}\ 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v\ x) = 5 \wedge \text{length}(\text{inouts}_v'\ x) = 2 \wedge [0, 0] = \text{inouts}_v'\ x) \wedge$
 $(\neg \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int}(\lceil \text{Rate} * \max(\text{inouts}_v\ x!(\text{Suc}\ 0))\ 0\rceil))))$
 $< \min\ 1(\text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}(\lceil \text{Rate} * \max(\text{inouts}_v\ 0!(\text{Suc}\ 0))\ 0\rceil)))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v\ x) = 5 \wedge$
 $\text{length}(\text{inouts}_v'\ x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v'\ x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v\ (n1 - \text{Suc}\ 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v\ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v\ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v\ (n1 - \text{Suc}\ 0)!3 = 0 \wedge \text{inouts}_v\ (n1 - \text{Suc}\ 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v\ x) = 5 \wedge \text{length}(\text{inouts}_v'\ x) = 2 \wedge [0, 0] = \text{inouts}_v'\ x) \wedge$
 $(\neg \text{hd}(\text{inouts}_v\ x) = 0 \longrightarrow$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}(\lceil \text{Rate} * \max(\text{inouts}_v\ x!(\text{Suc}\ 0))\ 0\rceil))) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v\ (n1 - \text{Suc}\ 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v\ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v\ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v\ (n1 - \text{Suc}\ 0)!3 = 0 \wedge \text{inouts}_v\ (n1 - \text{Suc}\ 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v\ x) = 5 \wedge \text{length}(\text{inouts}_v'\ x) = 2 \wedge [1, 1] = \text{inouts}_v'\ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v\ (n1 - \text{Suc}\ 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v\ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v\ n1!2 = 8$

$$\begin{aligned} & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \\ & \text{length}(\text{inouts}_v' x) = 2 \wedge \\ & [0, 0] = \text{inouts}_v' x \wedge \\ & (\text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\ & \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\ & (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0])) < 1 \longrightarrow \\ & (x = 0 \longrightarrow \\ & \text{length}(\text{inouts}_v 0) = 5 \wedge \\ & \text{length}(\text{inouts}_v' 0) = 2 \wedge \\ & [0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' \\ & 0) \wedge \\ & (0 < x \longrightarrow \\ & (\text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\ & (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\ & < \min 1 (\text{real-of-int} \\ & \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) + \\ & \quad 1 \longrightarrow \\ & (\neg \text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\ & \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\ & \quad \text{then } 0 \text{ else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\ & (\text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\ & \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\ & < \min 1 (\text{real-of-int} \\ & \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) + \\ & \quad 1 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \\ & \text{length}(\text{inouts}_v' x) = 2 \wedge \end{aligned}$$

$$\begin{aligned}
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& \quad (\neg \text{latch-rec-calc-output} \\
& \quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad x = \\
& \quad \quad 0 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& \quad \quad (\text{latch-rec-calc-output} \\
& \quad \quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \quad \quad \text{else } 1) \\
& \quad \quad \quad x = \\
& \quad \quad \quad 0 \longrightarrow \\
& \quad \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& \quad \quad \quad (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& \quad \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \quad \quad \quad \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& \quad \quad \quad [0, 0] = \text{inouts}_v' x \wedge \\
& \quad \quad \quad (\text{latch-rec-calc-output} \\
& \quad \quad \quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad \quad \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \quad \quad \quad \text{else } 1) \\
& \quad \quad \quad \quad x = \\
& \quad \quad \quad \quad 0 \longrightarrow \\
& \quad \quad \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)))))) \wedge \\
& (\neg \text{hd } (\text{inouts}_v 0) = 0 \longrightarrow \\
& \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& \quad (x = 0 \longrightarrow \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' 0) \wedge \\
& \quad (0 < x \longrightarrow \\
& \quad \quad (\text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& \quad \quad \quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\
& \quad \quad \quad < \min 0 (\text{real-of-int} \\
& \quad \quad \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) + \\
& \quad \quad \quad \quad 1 \longrightarrow \\
& \quad \quad \quad (\neg \text{latch-rec-calc-output} \\
& \quad \quad \quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8
\end{aligned}$$

$$\begin{aligned} & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\ & \quad \text{then } 0 \text{ else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\ & (\text{latch-rec-calc-output} \\ & \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{ inouts}_v n1!2 = 8 \\ & \quad \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\ & \quad < \text{min } 0(\text{real-of-int} \\ & \quad \quad (\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v 0!(\text{Suc } 0)) 0]))) + \\ & \quad 1 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \\ & \text{length}(\text{inouts}_v' x) = 2 \wedge \\ & [0, 0] = \text{inouts}_v' x \wedge \\ & (\text{latch-rec-calc-output} \\ & \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{ inouts}_v n1!2 = 8 \\ & \quad \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{hd}(\text{inouts}_v x) = 0 \longrightarrow \\ & (\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\ & (\neg \text{latch-rec-calc-output} \\ & \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{ inouts}_v n1!2 = 8 \\ & \quad \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\ & \quad \text{then } 0 \text{ else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\ & (\text{latch-rec-calc-output} \\ & \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\ & \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{ inouts}_v n1!2 = 8 \\ & \quad \quad \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & \quad \text{else } 1) \\ & x = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \\ & \text{length}(\text{inouts}_v' x) = 2 \wedge \\ & [0, 0] = \text{inouts}_v' x \wedge \end{aligned}$$

$(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) < 0 \longrightarrow$
 $(x = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v 0) = 5 \wedge$
 $\text{length}(\text{inouts}_v' 0) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v'$
 $0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } 0 (\text{real-of-int}$
 $\quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) +$
 $\quad 1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } 0 (\text{real-of-int}$
 $\quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) +$
 $\quad 1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$

$$\begin{aligned}
& (\neg \text{hd} (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)))))) \wedge \\
& (\neg x \leq \text{Suc } 0 \longrightarrow \\
& (\text{hd} (\text{inouts}_v (x - \text{Suc } 0)) = 0 \longrightarrow \\
& (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0 \rceil)) \\
& < \min (\text{vT-fd-sol-1} \\
& \quad (\lambda n1. \text{real-of-int} \\
& \quad \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v n1!(\text{Suc } 0)) 0 \rceil)))) \\
& \quad (\lambda n1. \text{if } \text{hd} (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } (\text{Suc } 0))) \\
& (\text{real-of-int} \\
& \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } (\text{Suc } 0))!(\text{Suc } 0)) 0 \rceil)))))) \\
& + \\
& 1 \longrightarrow \\
& (\text{hd} (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) \\
& < \min (\text{vT-fd-sol-1} \\
& \quad (\lambda n1. \text{real-of-int} \\
& \quad \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v n1!(\text{Suc } 0)) 0 \rceil)))) \\
& \quad (\lambda n1. \text{if } \text{hd} (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& (\text{real-of-int} \\
& \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0 \rceil)))))) + \\
& 1 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow
\end{aligned}$$

$$\begin{aligned}
& \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& x = \\
& 0 \wedge \\
& \text{latch-rec-calc-output} \\
& (\lambda n1. \text{ if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& (x - \text{Suc } 0) = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{ if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{ if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (x - \text{Suc } 0) = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)) \wedge \\
& (\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))) \\
& < \text{min } (vT\text{-fd-sol-1} \\
& \quad (\lambda n1. \text{ real-of-int} \\
& \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0)) \ 0 \rceil)))) \\
& \quad (\lambda n1. \text{ if } \text{hd } (\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& \quad (\text{real-of-int} \\
& \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ (x - \text{Suc } 0)!(\text{Suc } 0)) \ 0 \rceil)))) + \\
& \quad 1 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' \ x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' \ x \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{ if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{ if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow
\end{aligned}$$

$$\begin{aligned}
& \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (x - \text{Suc } 0) = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)) \wedge \\
& (\neg \text{hd}(\text{inouts}_v \ x) = 0 \longrightarrow \\
& (\text{int32} \ (\text{RoundZero} \ (\text{real-of-int} \ [\text{Rate} * \text{max} \ (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0])) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad x = \\
& \quad 0 \wedge \\
& \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad (x - \text{Suc } 0) = \\
& \quad 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (x - \text{Suc } 0) = \\
& \quad 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)) \wedge \\
& (\neg \text{int32} \ (\text{RoundZero} \ (\text{real-of-int} \ [\text{Rate} * \text{max} \ (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0])) < 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' \ x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' \ x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x =
\end{aligned}$$

$0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0])))$
 $< \text{min } (vT\text{-fd-sol-1}$
 $\quad (\lambda n1. \text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0)) 0])))$
 $\quad \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } (\text{Suc } 0)))$
 $\quad (\text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero}$
 $\quad \quad \quad (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } (\text{Suc } 0))!(\text{Suc } 0)) 0]))) +$
 $\quad \quad 1 \longrightarrow$
 $\quad (\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $\quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $\quad < \text{min } (vT\text{-fd-sol-1}$
 $\quad \quad (\lambda n1. \text{real-of-int}$
 $\quad \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0)) 0])))$
 $\quad \quad \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $\quad \quad (\text{real-of-int}$
 $\quad \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0]))) +$
 $\quad \quad \quad 1 \longrightarrow$
 $\quad (\neg \text{latch-rec-calc-output}$
 $\quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \quad \text{then } 0 \text{ else } 1)$
 $\quad \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \quad \text{then } 0 \text{ else } 1)$
 $\quad \quad x =$
 $\quad \quad 0 \longrightarrow$
 $\quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $\quad \quad (\text{latch-rec-calc-output}$
 $\quad \quad \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \quad \quad \text{then } 0 \text{ else } 1)$
 $\quad \quad \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \quad \quad \text{else } 1)$
 $\quad \quad \quad x =$
 $\quad \quad \quad 0 \longrightarrow$
 $\quad \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $\quad \quad \quad (\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $\quad \quad \quad < \text{min } (vT\text{-fd-sol-1}$
 $\quad \quad \quad \quad (\lambda n1. \text{real-of-int}$
 $\quad \quad \quad \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0)) 0])))$
 $\quad \quad \quad \quad \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $\quad \quad \quad \quad (\text{real-of-int}$
 $\quad \quad \quad \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0]))) +$

$$\begin{aligned}
& 1 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \text{else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)))) \wedge \\
& (\neg \text{hd } (\text{inouts}_v (x - \text{Suc } 0)) = 0 \longrightarrow \\
& (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\
& (\text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\
& < \text{min } (\text{vT-fd-sol-1} \\
& \quad (\lambda n1. \text{real-of-int} \\
& \quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0)) 0]))) \\
& \quad \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& \quad \text{real-of-int}
\end{aligned}$$

$$\begin{aligned}
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)))) + \\
& 1 \longrightarrow \\
(\neg \text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& x = \\
& 0 \wedge \\
\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& (x - Suc 0) = \\
& 0 \longrightarrow \\
& \text{length}(inouts_v x) = 5 \wedge \text{length}(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x \wedge \\
(\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(inouts_v x) = 5 \wedge \text{length}(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x \wedge \\
(\neg \text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (x - Suc 0) = \\
& 0 \longrightarrow \\
& \text{length}(inouts_v x) = 5 \wedge \text{length}(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge \\
(\neg \text{ real-of-int } (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)))) & \\
< \text{ min } (vT\text{-fd-sol-1} & \\
& (\lambda n1. \text{ real-of-int} \\
& \quad (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v n1!(Suc 0)) 0 \rceil)))) \\
& \quad (\lambda n1. \text{ if } \text{hd } (inouts_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - Suc 0)) \\
& \text{ (real-of-int} \\
& \quad (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)))) + \\
& 1 \longrightarrow \\
& \text{length}(inouts_v x) = 5 \wedge \\
& \text{length}(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
(\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x =
\end{aligned}$$

$0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad x =$
 $\quad 0 \wedge$
 $\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad x =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$

then 0 else 1)
($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
($\neg \text{latch-rec-calc-output}$
($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
then 0 else 1)
($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
then 0 else 1)
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
($\neg \text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \min(\text{vT-fd-sol-1}$
($\lambda n1. \text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0)) 0])))$
($\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0)(x - \text{Suc } 0))$
 $(\text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0]))) +$
 $1 \longrightarrow$
 $\neg \text{latch-rec-calc-output}$
($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
then 0 else 1)
($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
then 0 else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
then 0 else 1)
($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
($\neg \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \min(\text{vT-fd-sol-1}$
($\lambda n1. \text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0)) 0])))$
($\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0)(x - \text{Suc } 0))$
 $(\text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \max(\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0]))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$

$(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\quad (\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad x =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $\quad (\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad x =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge$
 $\quad \text{length}(\text{inouts}_v' x) = 2 \wedge$
 $\quad [0, 0] = \text{inouts}_v' x \wedge$
 $\quad (\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad x =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))))))$
assume a2: $\forall x. ?P x$

assume a3: $\text{inouts}_v x!3 = 1$
assume a4: $\text{inouts}_v x!2 = 4$
assume a5: $\text{inouts}_v x!0 = 1$
assume a6: $\text{inouts}_v (\text{Suc } x)!3 = 1$
assume a7: $\text{inouts}_v (\text{Suc } x)!2 = 8$
assume a8: $\text{inouts}_v (\text{Suc } x)!0 = 1$
assume a81: $\forall x. \text{hd } (\text{tl } (\text{inouts}_v' x)) = \text{inouts}_v x!(4)$
assume a9: $\forall xb \leq 200. \text{inouts}_v (\text{Suc } (\text{Suc } (x + xa + xb)))!0 = 0$
assume a10: $\forall xb \leq xa + 200. \text{inouts}_v (\text{Suc } (\text{Suc } (x + xb)))!3 = 1 \wedge \text{inouts}_v (\text{Suc } (\text{Suc } (x +$
 $xb)))!2 = 8$
assume a11: $\text{inouts}_v (\text{Suc } (x + xa))!0 = 1$

```

assume a12:  $\forall xb < xa. \text{hd}(\text{inouts}_v'(\text{Suc}(\text{Suc}(x + xb)))) = 0$ 
have len-inouts:  $\forall x. \text{length}(\text{inouts}_v x) = 5$ 
  using a2 by blast

have a11':  $\text{hd}(\text{inouts}_v(\text{Suc}(x + xa))) = 1$ 
  using a11 len-inouts
  by (metis hd-conv-nth list.size(3) zero-neq-numeral)

from a1 have a1':  $\forall x. \text{inouts}_v x!(\text{Suc } 0) = c\text{-door-open-time}$ 
  by simp
have 1:  $\forall x::\text{nat}. (\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0) 0 \rceil))) = 200)$ 
  using a1' by (simp add: RoundZero-def int32-def)
have 11:  $\forall x::\text{nat}. (\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0) 0 \rceil))))$ 
= 200)
  using a1' by (simp add: RoundZero-def int32-def)

have 12: (vT-fd-sol-1
  ( $\lambda n1. \text{real-of-int}$ 
    ( $\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0) 0 \rceil)))$ 
    ( $\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0$ )( $\text{Suc}(\text{Suc}(x + xa))$ ))) = 1
  proof -
    have 1: (vT-fd-sol-1
      ( $\lambda n1. \text{real-of-int}$ 
        ( $\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0) 0 \rceil)))$ 
        ( $\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0$ )( $\text{Suc}(\text{Suc}(x + xa))$ ))) =
        (vT-fd-sol-1
          ( $\lambda n1. 200$ )
          ( $\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0$ )( $\text{Suc}(\text{Suc}(x + xa))$ )))
        using 11 by simp
      then have 2: ... = 1
        apply (simp)
        using a9 a11 by (smt Nat.add-0-right a1 a2 hd-conv-nth le0 list.size(3) zero-less-Suc
zero-neq-numeral)
        show ?thesis
        using 1 2 by (simp)
    qed

have 13:  $\forall q < 200. (\text{vT-fd-sol-1}$ 
  ( $\lambda n1. \text{real-of-int}$ 
    ( $\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0) 0 \rceil)))$ 
    ( $\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0$ )( $\text{Suc}(\text{Suc}(x + xa + q))$ ))) =  $q + 1$ 
  apply (rule allI)
  proof -
    fix q::nat
    have 1:  $q < 200 \longrightarrow$ 
      (vT-fd-sol-1
        ( $\lambda n1. \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0) 0 \rceil)))$ 
        ( $\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0$ )( $\text{Suc}(\text{Suc}(x + xa + q))$ )))
        =  $\text{real}(q + 1)$ 
      proof (induct q)
        case 0
          then show ?case using 12 by simp
        next
          case (Suc q)
          then show ?case

```

```

apply (clarify)
apply (simp)
apply (rule conjI)
apply (clarify)
using 11 apply auto[1]
proof –
  assume a1:  $q < 199$ 
  have a1':  $Suc\ q < 200$ 
    using a1 by simp
  have 1:  $hd\ (inouts_v\ (Suc\ (Suc\ (Suc\ (x + xa + q)))) = (inouts_v\ (Suc\ (Suc\ (Suc\ (x + xa + q))))!0$ 
    using len-inouts
    by (metis Suc-numeral Zero-not-Suc hd-conv-nth list.size(3) semiring-norm(5))
  then have 2:  $\dots = (inouts_v\ (Suc\ (Suc\ (x + xa + Suc\ q))))!0$ 
    by (smt add-Suc-right)
  then have 3:  $\dots = 0$ 
  proof –
    show ?thesis
      using a1' a9 le-eq-less-or-eq by presburger
    qed
  show  $hd\ (inouts_v\ (Suc\ (Suc\ (Suc\ (x + xa + q)))) = 0$ 
    using 1 2 3 by linarith
  qed
qed
show  $q < 200 \longrightarrow vT\text{-fd-sol-1}$ 
  ( $\lambda n1.\ real\text{-of-int}\ (int32\ (RoundZero\ (real\text{-of-int}\ [Rate * \max\ (inouts_v\ n1!(Suc\ 0))\ 0])))$ )
  ( $\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0$ ) ( $Suc\ (Suc\ (x + xa + q))$ ) =  $real\ (q + 1)$ 
  using 1 by linarith
qed
have 130:  $\forall q < 200 . (vT\text{-fd-sol-1}\ (\lambda n1.\ 200))$ 
  ( $\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0$ ) ( $Suc\ (Suc\ (x + xa + q))$ ) =  $q + 1$ 
  using 13 by (simp add: 11)

have 14: ( $vT\text{-fd-sol-1}\ (\lambda n1.\ real\text{-of-int}\ (int32\ (RoundZero\ (real\text{-of-int}\ [Rate * \max\ (inouts_v\ n1!(Suc\ 0))\ 0])))$ )
  ( $\lambda n1.\ if\ hd\ (inouts_v\ n1) = 0\ then\ 1\ else\ 0$ ) ( $Suc\ (x + xa)$ ) = 0
  using a11 a11' 1 11 by (simp)

have output-at-x:  $hd\ (inouts_v'\ x) = 0$ 
  using a5 a2
  by (smt 1 hd-Cons-tl hd-conv-nth list.inject list.size(3) neq0-conv zero-neq-numeral)
have output-at-x-1:  $hd\ (inouts_v'\ (Suc\ x)) = 0$ 
  using a8 a2
  by (smt 1 hd-Cons-tl hd-conv-nth list.inject list.size(3) neq0-conv zero-neq-numeral)

have output-at-q:  $\forall q < 200 . hd\ (inouts_v'\ (Suc\ (Suc\ (x + xa + q)))) = 0$ 
  apply (rule allI)
  proof –
    fix q::nat
    have count-less:  $\forall q < 200.$ 
      ( $\neg\ real\text{-of-int}\ (int32\ (RoundZero\ (real\text{-of-int}\ [Rate * \max\ (inouts_v\ (Suc\ (Suc\ (x + xa + q))))!(Suc\ 0))\ 0])))$ )
      <  $\min\ (vT\text{-fd-sol-1}$ 

```

```

      (λn1. real-of-int
        (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc 0)) 0])))
      (λn1. if hd (inouts_v n1) = 0 then 1 else 0) (Suc (x + xa + q)))
    (real-of-int
      (int32 (RoundZero (real-of-int [Rate * max (inouts_v (Suc (x + xa + q))!(Suc 0))
0]))) +
      1)
  apply (rule allI)
  proof -
    fix q::nat
    show 1: q < 200 →
      ¬ real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v (Suc (Suc (x + xa +
q))!(Suc 0)) 0])))
      < min (vT-fd-sol-1
        (λn1. real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc 0))
0])))
        (λn1. if hd (inouts_v n1) = 0 then 1 else 0) (Suc (x + xa + q)))
      (real-of-int
        (int32 (RoundZero (real-of-int [Rate * max (inouts_v (Suc (x + xa + q))!(Suc 0))
0]))) +
        1)
    proof (induct q)
      case 0
      then show ?case
        using 1 11 14 a11 by simp
    next
      case (Suc q)
      then show ?case
        using 1 11 14 a11 13 by simp
    qed
  qed
  show q < 200 → hd (inouts_v' (Suc (Suc (x + xa + q)))) = 0
  proof (induct q)
    case 0
    then show ?case
      using a11 1 11 a2 13 count-less
      by (smt 14 Nat.add-0-right One-nat-def diff-Suc-1 list.sel(1) zero-less-Suc)
  next
    case (Suc q)
    then show ?case
      using count-less 1 11 a2
      by (smt One-nat-def Suc-lessD a1 diff-Suc-1 zero-less-Suc)
  qed
  qed
  have output-eq: ∀ x. hd (tl(inouts_v' x)) = hd(inouts_v' x)
    using a2 by (smt hd-Cons-tl list.inject not-gr0 tl-Nil)
  have input4-x: inouts_v (x)!4 = 0
    using output-at-x output-eq by (simp add: a81)
  have input4-x-1: inouts_v (Suc x)!4 = 0
    using output-at-x-1 output-eq by (simp add: a81)
  have input4-q: ∀ q<200. inouts_v (Suc (Suc (x + xa + q))!4 = 0
    using output-at-q a81 output-eq by auto
  have a12': ∀ xb<xa. (inouts_v (Suc (Suc (x + xb))))!4 = 0
    using a12 a81 using output-eq by auto

```

```

have input4-x-to-q:  $\forall q::\text{nat} . (q < xa \longrightarrow \text{inouts}_v (\text{Suc} (\text{Suc} (x + q)))!4 = 0) \wedge$ 
  ( $q \geq xa \wedge q < xa + 200 \longrightarrow \text{inouts}_v (\text{Suc} (\text{Suc} (x + q)))!4 = 0$ )
using input4-q a12' apply (simp)
apply (rule allI, clarify)
by (metis (full-types) add-less-cancel-left le-Suc-ex semiring-normalization-rules(25))

have latch-m-1: latch-rec-calc-output
  ( $\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$ 
     $\text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$ 
    then 0 else 1)
  ( $\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ 
    then 0 else 1)
  (Suc x) = 1
apply (simp)
using a3 a4 a5 a6 a7 a8
by (metis hd-conv-nth input4-x len-inouts list.size(3) zero-neq-numeral zero-neq-one)

have latch-1-q-200:  $\forall q \leq (xa + 200) . \text{latch-rec-calc-output}$ 
  ( $\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$ 
     $\text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$ 
    then 0 else 1)
  ( $\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ 
    then 0 else 1)
  (Suc (Suc (x + q))) = 1
apply (rule allI)
proof –
  fix q::nat
  show  $q \leq xa + 200 \longrightarrow$ 
    latch-rec-calc-output
    ( $\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!(2)$ 
    = 8 then 0
      else 1)
    ( $\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$  then 0
      else 1)
    (Suc (Suc (x + q))) = 1
  proof (induct q)
  case 0
  then show ?case
    using a6 input4-x-1 latch-m-1 by auto
  next
  case (Suc q)
  then show ?case
  proof –
    assume a1:  $q \leq xa + 200 \longrightarrow$ 
      latch-rec-calc-output
      ( $\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$ 
      n1!(2) = 8 then 0
        else 1)
      ( $\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$ 
      then 0 else 1)
      (Suc (Suc (x + q))) = 1
    have 1:  $\text{Suc } q \leq xa + 200 \longrightarrow$ 
      (latch-rec-calc-output
      ( $\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$ 

```

$n1!(2) = 8$ then 0
 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$
 then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + \text{Suc } q))$)) = (*latch-rec-calc-output*
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!(2) = 8$ then 0
 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$
 then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + q))$))
apply (*clarify*)
proof –
assume $a1: \text{Suc } q \leq xa + 200$
have 1: ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow$
 $\text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!(2) = 8$ then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + \text{Suc } q)) = 0$
 using $a10$ $a1$ **by** *auto*
have 2: ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge$
 $\text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$ then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + \text{Suc } q)) = 0$
 apply (*simp*)
 apply (*rule conjI*)
 using $a10$ **apply** (*smt Suc-leD a1*)
 using *input4-x-to-q a1*
 by (*metis Suc-le-eq le-eq-less-or-eq nat-le-linear*)
show ((*latch-rec-calc-output*
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg$
 $\text{inouts}_v n1!(2) = 8$ then 0
 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$
 then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + \text{Suc } q))$)) = (*latch-rec-calc-output*
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg$
 $\text{inouts}_v n1!(2) = 8$ then 0
 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$
 then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + q))$))
 using 1 2 **by** (*smt add-Suc-right latch-rec-calc-output.simps(2)*)
qed

show $\text{Suc } q \leq xa + 200 \longrightarrow$
 latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!(2) = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!(2) = 8$ then 0
 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!(3) = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!(4) = 0$
 then 0 else 1)
 ($\text{Suc } (\text{Suc } (x + \text{Suc } q)) = 1$
 using 1 $a1$ **by** *linarith*
qed
qed
qed
have *latch-at-202*:

latch-rec-calc-output

($\lambda n1$. if $inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 then 0 else 1)
($\lambda n1$. if $n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0$
 then 0 else 1) (202 + (x + xa)) = 1

proof –

have 1: *latch-rec-calc-output*

($\lambda n1$. if $inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 then 0 else 1)
($\lambda n1$. if $n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0$
 then 0 else 1)
($Suc (Suc (x+xa+200))$) = 1

using *latch-1-q-200*

by (*metis (no-types, lifting) add.assoc add-le-cancel-left add-less-cancel-left mono-nat-linear-lb*)

have 2: ($Suc (Suc (x+xa+200))$) = (202 + (x + xa))

by *auto*

show ?thesis

using 1 2 **by** *simp*

qed

have *count-at-198*:

vT-fd-sol-1 ($\lambda n1$. 200) ($\lambda n1$. if $hd (inouts_v\ n1) = 0$ then 1 else 0) (200 + (x + xa)) = 199

proof –

have 1: *vT-fd-sol-1* ($\lambda n1$. 200) ($\lambda n1$. if $hd (inouts_v\ n1) = 0$ then 1 else 0)

($Suc (Suc (x + xa + 198))$) = 199

using 130 **by** (*metis (no-types, lifting) Suc-numeral less-add-Suc2 numeral-Bit0 numeral-Bit1*
of-nat-numeral one-plus-numeral semiring-norm(3) semiring-norm(5) semiring-norm(8))

have 2: (200 + (x + xa)) = ($Suc (Suc (x + xa + 198))$)

by *auto*

show ?thesis

using 1 2 **by** *presburger*

qed

have *count-at-199*:

vT-fd-sol-1 ($\lambda n1$. 200) ($\lambda n1$. if $hd (inouts_v\ n1) = 0$ then 1 else 0) (201 + (x + xa)) = 200

proof –

have 1: *vT-fd-sol-1* ($\lambda n1$. 200) ($\lambda n1$. if $hd (inouts_v\ n1) = 0$ then 1 else 0)

($Suc (Suc (x + xa + 199))$) = 200

using 130

by (*metis Suc-numeral lessI numeral-plus-one of-nat-numeral semiring-norm(5) semiring-norm(8)*)

have 2: (201 + (x + xa)) = ($Suc (Suc (x + xa + 199))$)

by *auto*

show ?thesis

using 1 2 **by** *presburger*

qed

have $inouts_v (Suc (Suc (x + xa + 199)))!0 = 0$

using *a9 len-inouts*

by (*metis Suc-numeral le-eq-less-or-eq lessI semiring-norm(5) semiring-norm(8)*)

then have $hd(inouts_v (Suc (Suc (x + xa + 199)))) = 0$

using *a9 len-inouts* **by** (*smt hd-conv-nth list.size(3) zero-neq-numeral*)

then have *a9-199*: $hd (inouts_v (201 + (x + xa))) = 0$

by (*simp add: semiring-normalization-rules(25)*)

```

have a9-200-0: inoutsv (Suc (Suc (x + xa + 200)))!0 = 0
  using a9 len-inouts by blast
then have hd(inoutsv (Suc (Suc (x + xa + 200)))) = 0
  using a9 len-inouts by (smt hd-conv-nth list.size(3) zero-neq-numeral)
then have a9-200: hd(inoutsv (202 + (x + xa))) = 0
  by (simp add: semiring-normalization-rules(25))
have output-at-p-200-imply: (?P (Suc (Suc (x + xa + 200)))) → (inoutsv' (202 + (x + xa))) =
[1,1]
  apply (simp)
  apply (simp add: a9-199)
  apply (simp add: 1 11)
  apply (simp add: count-at-198)
  apply (simp add: a9-200)
  apply (simp add: count-at-199)
  by (simp add: latch-at-202)
have output-at-p-200: (?P (Suc (Suc (x + xa + 200))))
  using a2 by smt
show inoutsv' (202 + (x + xa)) = [1,1]
  using output-at-p-200 output-at-p-200-imply by fastforce
qed

```

Secondly to verify the refinement relation for the feedback.

```

lemma req-01-ref: req-01-1-contract fD (4, 1) ⊆ plf-rise1shot-simp fD (4, 1)
  apply (rule feedback-mono[of 5 2])
  using SimBlock-req-01-1-contract apply (blast)
  using post-landing-finalize-1-simblock apply (blast)
  using req-01-ref-plf-rise1shot apply (blast)
  by (auto)

```

Thirdly to verify the requirement contract satisfied by the feedback of *req-01-1-contract*.

```

lemma req-01-fd-ref:
  req-01-contract ⊆ req-01-1-contract fD (4, 1)
  using inps-req-01-1-contract outps-req-01-1-contract apply (simp add: PreFD-def PostFD-def)
  proof -
    show req-01-contract ⊆ (∃ x • (true ⊢n
      (∀ n • #u($inouts(«n»)a) =u «4» ∧ #u($inouts'«n»)a) =u «5» ∧ $inouts'«n»a) =u
«f-PreFD x 4»($inouts)a(«n»a)) ; ;
      req-01-1-contract ; ;
      (true ⊢n
        (∀ n • #u($inouts(«n»)a) =u «2» ∧
          #u($inouts'«n»)a) =u «Suc 0» ∧
          $inouts'«n»a) =u «f-PostFD (Suc 0)»($inouts)a(«n»a) ∧ «uapply»($inouts(«n»)a)(«Suc
0») =u «x n»)))
    apply (simp (no-asm) add: req-01-1-contract-def req-01-contract-def)

  apply (rel-simp)
  apply (simp add: f-PostFD-def f-PreFD-def)
  proof -
    fix okv::bool and inoutsv::nat⇒real list and
      okv'::bool and inoutsv'::nat⇒real list and x::nat⇒real and
      okv''::bool and inoutsv''::nat⇒real list and okv'''::bool and
      inoutsv'''::nat⇒real list
    assume a1: (∀ xa. (hd (inoutsv xa • [x xa]) = 0 ∨ hd (inoutsv xa • [x xa]) = 1) ∧
      (inoutsv xa • [x xa])!(Suc 0) = c-door-open-time ∧

```

$$\begin{aligned}
& ((inouts_v \ xa \bullet [x \ xa])!3 = 0 \vee (inouts_v \ xa \bullet [x \ xa])!3 = 1) \longrightarrow \\
& ok_v''' \wedge \\
& (\forall x. \text{length}(inouts_v''' \ x) = 2) \wedge \\
& (\forall xa. (inouts_v \ xa \bullet [x \ xa])!3 = 1 \wedge \\
& \quad (inouts_v \ xa \bullet [x \ xa])!2 = 4 \wedge \\
& \quad (inouts_v \ xa \bullet [x \ xa])!0 = 1 \wedge \\
& \quad (inouts_v \ (Suc \ xa) \bullet [x \ (Suc \ xa)])!3 = 1 \wedge \\
& \quad (inouts_v \ (Suc \ xa) \bullet [x \ (Suc \ xa)])!2 = 8 \wedge \\
& \quad (inouts_v \ (Suc \ xa) \bullet [x \ (Suc \ xa)])!0 = 1 \wedge (\forall xa. \text{hd} \ (\text{tl}(inouts_v''' \ xa)) = (inouts_v \ xa \bullet [x \\
& xa])!4) \longrightarrow \\
& (\forall xb. (\forall xc \leq 200. (inouts_v \ (Suc \ (Suc \ (xa + xb + xc))) \bullet [x \ (Suc \ (Suc \ (xa + xb + xc))]))!0 \\
& = 0) \wedge \\
& \quad (\forall xc \leq xb + 200. \\
& \quad \quad (inouts_v \ (Suc \ (Suc \ (xa + xc))) \bullet [x \ (Suc \ (Suc \ (xa + xc))]))!3 = 1 \wedge \\
& \quad \quad (inouts_v \ (Suc \ (Suc \ (xa + xc))) \bullet [x \ (Suc \ (Suc \ (xa + xc))]))!2 = 8) \wedge \\
& \quad \quad (inouts_v \ (Suc \ (xa + xb)) \bullet [x \ (Suc \ (xa + xb))])!0 = 1 \wedge \\
& \quad \quad (\forall xc < xb. \text{hd} \ (inouts_v''' \ (Suc \ (Suc \ (xa + xc)))) = 0) \longrightarrow \\
& \quad \quad inouts_v''' \ (202 + (xa + xb)) = [1, 1])) \\
\text{assume } a2: & ok_v''' \longrightarrow \\
& ok_v' \wedge \\
& (\forall xa. \text{length}(inouts_v''' \ xa) = 2 \wedge \\
& \quad \text{length}(inouts_v' \ xa) = Suc \ 0 \wedge \\
& \quad inouts_v' \ xa = \text{take} \ (Suc \ 0) \ (inouts_v''' \ xa) \bullet \text{drop} \ (Suc \ (Suc \ 0)) \ (inouts_v''' \ xa) \\
& \quad \wedge inouts_v''' \ xa!(Suc \ 0) = x \ xa) \\
\text{assume } a3: & \forall x. (\text{hd} \ (inouts_v \ x) = 0 \vee \text{hd} \ (inouts_v \ x) = 1) \wedge \\
& inouts_v \ x!(Suc \ 0) = c\text{-door-open-time} \wedge (inouts_v \ x!3 = 0 \vee inouts_v \ x!3 = 1) \\
\text{assume } a4: & \forall xa. \text{length}(inouts_v \ xa) = 4 \wedge \text{length}(inouts_v'' \ xa) = 5 \wedge \\
& inouts_v'' \ xa = \text{take} \ 4 \ (inouts_v \ xa) \bullet x \ xa \ \# \ \text{drop} \ 4 \ (inouts_v \ xa) \\
\text{from } a4 \ \text{have } 1: & \forall xa. \text{length}(inouts_v \ xa) = 4 \\
& \text{by } \text{blast} \\
\text{have } 2: & (\forall xa. (((\text{hd} \ (inouts_v \ xa \bullet [x \ xa]) = 0 \vee \text{hd} \ (inouts_v \ xa \bullet [x \ xa]) = 1) \wedge \\
& \quad (inouts_v \ xa \bullet [x \ xa])!(Suc \ 0) = c\text{-door-open-time} \wedge \\
& \quad ((inouts_v \ xa \bullet [x \ xa])!3 = 0 \vee (inouts_v \ xa \bullet [x \ xa])!3 = 1))) \\
& = ((\text{hd} \ (inouts_v \ xa) = 0 \vee \text{hd} \ (inouts_v \ xa) = 1) \wedge \\
& \quad inouts_v \ xa!(Suc \ 0) = c\text{-door-open-time} \wedge (inouts_v \ xa!3 = 0 \vee inouts_v \ xa!3 = 1)))) \\
& \text{using } 1 \\
& \text{by} \ (\text{metis} \ Suc\text{-mono} \ Suc\text{-numeral} \ \text{hd-append2} \ \text{length-greater-0-conv} \ \text{nth-append} \ \text{numeral-2-eq-2} \\
& \quad \text{numeral-3-eq-3} \ \text{semiring-norm}(2) \ \text{semiring-norm}(8) \ \text{zero-less-Suc}) \\
\text{have } 3: & ok_v''' \\
& \text{using } 2 \ a3 \ a1 \ \text{by} \ \text{simp} \\
\text{have } 4: & ok_v' \\
& \text{using } a2 \ 3 \ \text{by} \ \text{blast} \\
\text{have } 5: & \forall xa. inouts_v' \ xa = [\text{hd} \ (inouts_v''' \ xa)] \\
\text{using } 3 \ a2 \ \text{by} & \ (\text{metis} \ \text{append-eq-conv-conj} \ \text{length-Cons} \ \text{list.size}(3) \ \text{list-equal-size2} \ \text{self-append-conv}) \\
\text{have } 6: & \forall xa. inouts_v''' \ xa!(Suc \ 0) = x \ xa \\
& \text{using } a2 \ 3 \ \text{by} \ \text{blast} \\
\text{have } \text{input-at-3}: & \forall xa. (inouts_v \ xa \bullet [x \ xa])!3 = inouts_v \ xa!3 \\
& \text{using } 1 \ \text{by} \ (\text{simp} \ \text{add:} \ \text{nth-append}) \\
\text{have } \text{input-at-2}: & \forall xa. (inouts_v \ xa \bullet [x \ xa])!2 = inouts_v \ xa!2 \\
& \text{using } 1 \ \text{by} \ (\text{simp} \ \text{add:} \ \text{nth-append}) \\
\text{have } \text{input-at-1}: & \forall xa. (inouts_v \ xa \bullet [x \ xa])!1 = inouts_v \ xa!1 \\
& \text{using } 1 \ \text{by} \ (\text{simp} \ \text{add:} \ \text{nth-append}) \\
\text{have } \text{input-at-0}: & \forall xa. (inouts_v \ xa \bullet [x \ xa])!0 = inouts_v \ xa!0 \\
& \text{using } 1 \ \text{by} \ (\text{simp} \ \text{add:} \ \text{nth-append}) \\
\text{have } \text{input-at-4}: & \forall xa. (inouts_v \ xa \bullet [x \ xa])!4 = x \ xa
\end{aligned}$$

```

using 1 by (simp add: nth-append)
have feedback: (∀ xa. hd (tl(inouts_v''' xa)) = (inouts_v xa • [x xa])!4) =
  (∀ xa. (inouts_v''' xa)!(Suc 0) = (x xa))
by (metis 3 One-nat-def a2 diff-Suc-1 hd-conv-nth input-at-4 length-greater-0-conv
  length-tl nth-tl numeral-2-eq-2 zero-less-one)
have a1':
  (∀ x. length(inouts_v''' x) = 2) ∧
  (∀ xa. (inouts_v xa)!3 = 1 ∧
    (inouts_v xa)!2 = 4 ∧
    (inouts_v xa)!0 = 1 ∧
    (inouts_v (Suc xa))!3 = 1 ∧
    (inouts_v (Suc xa))!2 = 8 ∧
    (inouts_v (Suc xa))!0 = 1 ∧ (∀ xa. (inouts_v''' xa)!(Suc 0) = (x xa)) →
    (∀ xb. (∀ xc ≤ 200. (inouts_v (Suc (Suc (xa + xb + xc))))!0 = 0) ∧
      (∀ xc ≤ xb + 200.
        (inouts_v (Suc (Suc (xa + xc))))!3 = 1 ∧
        (inouts_v (Suc (Suc (xa + xc))))!2 = 8) ∧
        (inouts_v (Suc (xa + xb)))!0 = 1 ∧
        (∀ xc < xb. hd (inouts_v''' (Suc (Suc (xa + xc)))) = 0) →
        inouts_v''' (202 + (xa + xb)) = [1, 1]))
    using input-at-0 input-at-1 input-at-2 input-at-3 input-at-4 a1 6 2 3 a3 feedback
    by simp
  show ok_v' ∧
    (∀ x. length(inouts_v' x) = Suc 0) ∧
    (∀ x. inouts_v x!3 = 1 ∧
      inouts_v x!2 = 4 ∧ inouts_v x!0 = 1 ∧ inouts_v (Suc x)!3 = 1 ∧
      inouts_v (Suc x)!2 = 8 ∧ inouts_v (Suc x)!0 = 1 →
      (∀ xa. (∀ xb ≤ 200. inouts_v (Suc (Suc (x + xa + xb)))!0 = 0) ∧
        (∀ xb ≤ xa + 200. inouts_v (Suc (Suc (x + xb)))!3 = 1 ∧ inouts_v (Suc (Suc (x +
xb)))!2 = 8) ∧
        inouts_v (Suc (x + xa))!0 = 1 ∧ (∀ xb < xa. hd (inouts_v' (Suc (Suc (x + xb)))) = 0)
      →
        inouts_v' (202 + (x + xa)) = [1]))
    apply (rule conjI)
    using 4 apply (simp)
    apply (rule conjI)
    using 3 a2 apply blast
    apply (rule allI, clarify)
    using a1' apply (auto)
    by (simp add: 5 6)
  qed
qed

```

Finally, the requirement is held for the *post-landing-finalize-1* because of transitivity of refinement relation.

lemma req-01:

```

req-01-contract ⊆ post-landing-finalize-1
apply (simp only: post-landing-finalize-1-simp)
using req-01-fd-ref req-01-ref by auto

```

C.5.2 Requirement 02

post-landing-finalize-req-02: A finalize event is broadcast only once while the aircraft is on the ground.

req-02-contract is the requirement to be verified. Its precondition is the same as *req-01-contract*. Its postcondition specifies that

- it always has four inputs and one output;
- the requirement:
 - if a finalize event has been broadcast at step m ,
 - while the aircraft is on ground: *ac-on-ground* is true and $mode = GROUND$,
 - then a finalize event won't be broadcast again.

definition *req-02-contract* $\equiv ((\forall n::nat \cdot ($
 $\langle\langle \lambda x n.$
 $($
 $(hd(x n) = 0 \vee hd(x n) = 1) \wedge (* door-closed is boolean *)$
 $((x n)!1 = c-door-open-time) \wedge (* door-open-time *)$
 $((x n)!3 = 0 \vee (x n)!3 = 1) (* ac-on-ground is boolean *)$
 $\rangle\rangle) (\&inouts)_a (\langle\langle n \rangle\rangle)_a :: sim-state upred)$
 \vdash_n
 $((\forall n::nat \cdot$
 $((\#_u(\$inouts (\langle\langle n \rangle\rangle)_a)) =_u \langle\langle 4 \rangle\rangle) \wedge$
 $((\#_u(\$inouts' (\langle\langle n \rangle\rangle)_a)) =_u \langle\langle 1 \rangle\rangle) \wedge$
 $(* m : finalize-event$
 $... : mode is GROUND and ac-on-ground is true$
 $p : mode is GROUND and ac-on-ground is true \Rightarrow \neg finalize-event$
 $*)$
 $(\forall m::nat \cdot$
 $($
 $(head_u(\$inouts' (\langle\langle m \rangle\rangle)_a) =_u 1) (* finalize-event at m *)$
 \Rightarrow
 $($
 $\forall p::nat \cdot$
 $($
 $(\forall q::nat \cdot ((\langle\langle q \rangle\rangle \leq_u \langle\langle p \rangle\rangle) \Rightarrow$
 $((\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+1+q \rangle\rangle)_a)_a (\mathcal{I})_a =_u 1) (* ac-on-ground = true *) \wedge$
 $(\langle\langle nth \rangle\rangle (\$inouts (\langle\langle m+1+q \rangle\rangle)_a)_a (\mathcal{I})_a =_u 8) (* mode = GROUND *)))$
 $) (* the aircraft is always on the ground from m+1 to m+1+p *)$
 $\Rightarrow (\$inouts' (\langle\langle m+1+p \rangle\rangle)_a) =_u \langle\langle 0 \rangle\rangle (* then the finalize-event is false. *)$
 $)$
 $)$
 $))$
 $))$
 $))$

req-02-1-contract is the contract for *post-landing-finalize-1* without feedback: *plf-rise1shot-simp*. It is similar to *req-02-contract* except that 1) it has five inputs and two outputs (the feedback operator will remove one input and one output); 2) the 2nd output is equal to the 4th input since they are connected together by the feedback loop.

definition *req-02-1-contract* $\equiv ((\forall n::nat \cdot ($
 $\langle\langle \lambda x n.$
 $($
 $(hd(x n) = 0 \vee hd(x n) = 1) \wedge (* door-closed is boolean *)$
 $((x n)!1 = c-door-open-time) \wedge (* door-open-time *)$
 $((x n)!3 = 0 \vee (x n)!3 = 1) (* ac-on-ground is boolean *)$
 $\rangle\rangle) (\&inouts)_a (\langle\langle n \rangle\rangle)_a :: sim-state upred)$
 \vdash_n

```

((∀ n::nat ·
  ((#u($inouts («n»)a) =u «5») ∧
  ((#u($inouts' («n»)a) =u «2»)) ∧
  (* m : finalize-event
  ... : mode is GROUND and ac-on-ground is true
  p : mode is GROUND and ac-on-ground is true ⇒ ¬finalize-event
  *)
(∀ m::nat ·
  (
    (headu($inouts' («m»)a) =u 1) (* finalize-event at m *) ∧
    (∀ n::nat · (headu(tailu($inouts' («n»)a) =u «nth» ($inouts («n»)a) (4)a))
    ⇒
    (
      ∀ p::nat ·
      (
        (∀ q::nat · ((«q» ≤u «p»)) ⇒
          ((«nth» ($inouts («m+1+q»)a)a (3)a =u 1) (* ac-on-ground = true *) ∧
            («nth» ($inouts («m+1+q»)a)a (2)a =u 8) (* mode = GROUND *)))
        ) (* the aircraft is always on the ground from m+1 to m+1+p *)
        ⇒ ($inouts' («m+1+p»)a) =u ⟨0,0⟩(* then the finalize-event is false. *)
      )
    )
  )
  ))))

```

lemma *SimBlock-req-02-1-contract:*

SimBlock 5 2 req-02-1-contract

apply (*simp add: SimBlock-def req-02-1-contract-def*)

apply (*rel-auto*)

apply (*rule-tac x = λna. [1, 20, if na = 1 then 8 else 4, 1, 0] in exI*)

apply (*rule conjI, simp*)

apply (*rule-tac x = λna. [0, 0] in exI*)

by (*simp*)

lemma *inps-req-02-1-contract:*

inps req-02-1-contract = 5

using *SimBlock-req-02-1-contract inps-P by blast*

lemma *outps-req-02-1-contract:*

outps req-02-1-contract = 2

using *SimBlock-req-02-1-contract outps-P by blast*

In order to verify this requirement, firstly to verify the contract *req-02-1-contract* refined by *plf-rise1shot-simp*.

lemma *req-02-ref-plf-rise1shot: req-02-1-contract ⊆ plf-rise1shot-simp*

apply (*simp add: FBlock-def plf-rise1shot-simp-def req-02-1-contract-def*)

apply (*rule ndesign-refine-intro*)

apply *simp*

apply (*unfold upred-defs urel-defs*)

apply (*simp add: fun-eq-iff relcomp-unfold OO-def*

lens-defs upred-defs alpha-splits Product-Type.split-beta)?

apply (*transfer*)

apply (*simp; safe*)

proof –

fix *inouts_v inouts'_v::nat ⇒ real list and x::nat and xa::nat*

assume *a1: ∀ x. (hd (inouts_v x) = 0 ∨ hd (inouts_v x) = 1) ∧*

$inouts_v\ x!(Suc\ 0) = c\text{-door-open-time} \wedge (inouts_v\ x!3 = 0 \vee inouts_v\ x!3 = 1)$
let $?P = \lambda x. (x \leq Suc\ 0 \longrightarrow$
 $(hd\ (inouts_v\ 0) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0])) < 1 \longrightarrow$
 $(x = 0 \longrightarrow length(inouts_v\ 0) = 5 \wedge length(inouts_v'\ 0) = 2 \wedge [0, 0] = inouts_v'\ 0) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])))$
 $< min\ 1\ (real-of-int$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0]))) +$
 $1 \longrightarrow$
 $(\neg\ latch\text{-rec-calc-output}$
 $(\lambda n1. if\ inouts_v\ (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg\ inouts_v\ n1!2 = 8$
 $then\ 0\ else\ 1)$
 $(\lambda n1. if\ n1 = 0 \vee \neg\ inouts_v\ (n1 - Suc\ 0)!3 = 0 \wedge inouts_v\ (n1 - Suc\ 0)!4 = 0$
 $then\ 0\ else\ 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [1, 1] = inouts_v'\ x) \wedge$
 $(latch\text{-rec-calc-output}$
 $(\lambda n1. if\ inouts_v\ (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg\ inouts_v\ n1!2 = 8$
 $then\ 0\ else\ 1)$
 $(\lambda n1. if\ n1 = 0 \vee \neg\ inouts_v\ (n1 - Suc\ 0)!3 = 0 \wedge inouts_v\ (n1 - Suc\ 0)!4 = 0\ then\ 0$
 $else\ 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [0, 0] = inouts_v'\ x)) \wedge$
 $(\neg\ real-of-int\ (int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])))$
 $< min\ 1\ (real-of-int$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0]))) +$
 $1 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v'\ x) = 2 \wedge$
 $[0, 0] = inouts_v'\ x \wedge$
 $(latch\text{-rec-calc-output}$
 $(\lambda n1. if\ inouts_v\ (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg\ inouts_v\ n1!2 = 8$
 $then\ 0\ else\ 1)$
 $(\lambda n1. if\ n1 = 0 \vee \neg\ inouts_v\ (n1 - Suc\ 0)!3 = 0 \wedge inouts_v\ (n1 - Suc\ 0)!4 = 0\ then\ 0$
 $else\ 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [0, 0] = inouts_v'\ x))) \wedge$
 $(\neg\ hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(int32\ (RoundZero\ (real-of-int\ [Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $(\neg\ latch\text{-rec-calc-output}$
 $(\lambda n1. if\ inouts_v\ (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg\ inouts_v\ n1!2 = 8$
 $then\ 0\ else\ 1)$
 $(\lambda n1. if\ n1 = 0 \vee \neg\ inouts_v\ (n1 - Suc\ 0)!3 = 0 \wedge inouts_v\ (n1 - Suc\ 0)!4 = 0$
 $then\ 0\ else\ 1)$
 $x =$
 $0 \longrightarrow$

$length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $\quad hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0 \rceil)) < 0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $\quad hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0 \rceil)) < 1 \longrightarrow$
 $(x = 0 \longrightarrow$
 $length(inouts_v\ 0) = 5 \wedge$
 $length(inouts_v'\ 0) = 2 \wedge$
 $[0, 0] = inouts_v'\ 0 \wedge length(inouts_v\ 0) = 5 \wedge length(inouts_v'\ 0) = 2 \wedge [0, 0] = inouts_v'$
 $0) \wedge$
 $(0 < x \longrightarrow$
 $(hd\ (inouts_v\ x) = 0 \longrightarrow$
 $(real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0 \rceil)))$
 $< min\ 1\ (real-of-int$
 $\quad (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ 0!(Suc\ 0))\ 0 \rceil)))) +$
 $\quad 1 \longrightarrow$
(\neg latch-rec-calc-output
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $\quad hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow$
 $\quad hd\ (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v\ n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg real-of-int\ (int32\ (RoundZero\ (real-of-int\ \lceil Rate * max\ (inouts_v\ x!(Suc\ 0))\ 0 \rceil)))$
 $< min\ 1\ (real-of-int$

$$\begin{aligned}
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v 0!(Suc 0)) 0 \rceil))) + \\
& 1 \longrightarrow \\
& length(inouts_v x) = 5 \wedge \\
& length(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad then 0 else 1) \\
& (\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0 \\
& \quad else 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge \\
& (\neg hd (inouts_v x) = 0 \longrightarrow \\
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& (\neg latch-rec-calc-output \\
& (\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad then 0 else 1) \\
& (\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad then 0 else 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad then 0 else 1) \\
& (\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0 \\
& \quad else 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge \\
& (\neg int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge \\
& length(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad then 0 else 1) \\
& (\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0 \\
& \quad else 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)))))) \wedge \\
& (\neg hd (inouts_v 0) = 0 \longrightarrow \\
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v 0!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& (x = 0 \longrightarrow length(inouts_v 0) = 5 \wedge length(inouts_v' 0) = 2 \wedge [0, 0] = inouts_v' 0) \wedge \\
& (0 < x \longrightarrow \\
& (hd (inouts_v x) = 0 \longrightarrow \\
& (real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil))) \\
& < min 0 (real-of-int \\
& \quad (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v 0!(Suc 0)) 0 \rceil))) +
\end{aligned}$$

$1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $\quad < \text{min } 0 (\text{real-of-int}$
 $\quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0)) 0]))) +$
 $\quad 1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0]))) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$

$(\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $(x = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v 0) = 5 \wedge$
 $\text{length}(\text{inouts}_v' 0) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v'$
 $0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd} (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)))$
 $< \text{min } 0 (\text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)))$
 $< \text{min } 0 (\text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$

else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)))))) \wedge$
 $(\neg x \leq \text{Suc } 0 \longrightarrow$
 $(\text{hd}(\text{inouts}_v (x - \text{Suc } 0)) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0 \rceil))$
 $< \min(\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0)) 0 \rceil))))$
 $(\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0)(x - \text{Suc}(\text{Suc } 0)))$
 $(\text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v (x - \text{Suc}(\text{Suc } 0))!(\text{Suc } 0)) 0 \rceil))))$
 $+$
 $1 \longrightarrow$
 $(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))$
 $< \min(\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int}$
 $(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v n1!(\text{Suc } 0)) 0 \rceil))))$
 $(\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0)(x - \text{Suc } 0))$
 $(\text{real-of-int}$

$$\begin{aligned}
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)))) + \\
& 1 \longrightarrow \\
(\neg \text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& x = \\
& 0 \wedge \\
\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& (x - Suc 0) = \\
& 0 \longrightarrow \\
& \text{ length}(inouts_v x) = 5 \wedge \text{ length}(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge \\
(\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{ length}(inouts_v x) = 5 \wedge \text{ length}(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge \\
(\neg \text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (x - Suc 0) = \\
& 0 \longrightarrow \\
& \text{ length}(inouts_v x) = 5 \wedge \text{ length}(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge \\
(\neg \text{ real-of-int } (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)))) & \\
< \text{ min } (vT\text{-fd-sol-1} & \\
& (\lambda n1. \text{ real-of-int} \\
& \quad (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v n1!(Suc 0)) 0 \rceil)))) \\
& \quad (\lambda n1. \text{ if } \text{hd } (inouts_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - Suc 0)) \\
& \text{ (real-of-int} \\
& \quad (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)))) + \\
& 1 \longrightarrow \\
& \text{ length}(inouts_v x) = 5 \wedge \\
& \text{ length}(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
(\text{ latch-rec-calc-output} & \\
& (\lambda n1. \text{ if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad \text{hd } (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x =
\end{aligned}$$

$0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad x =$
 $\quad 0 \wedge$
 $\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \quad \text{else } 1)$
 $\quad x =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \quad \text{then } 0 \text{ else } 1)$
 $\quad (x - \text{Suc } 0) =$
 $\quad 0 \longrightarrow$
 $\quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $\quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$

$$\begin{aligned}
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{ inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{ inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad (x - \text{Suc } 0) = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge \\
& (\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) \\
0]))) \\
& < \text{min} (vT-fd-sol-1 \\
& \quad (\lambda n1. \text{real-of-int} \\
& \quad \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v n1!(\text{Suc } 0)) 0]))) \\
& \quad \quad (\lambda n1. \text{ if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc} (\text{Suc } 0))) \\
& \quad (\text{real-of-int} \\
& \quad \quad (\text{int32} (\text{RoundZero} \\
& \quad \quad \quad (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc} (\text{Suc } 0)!(\text{Suc } 0)) 0]))) + \\
& \quad \quad 1 \longrightarrow \\
& \quad (\text{hd}(\text{inouts}_v x) = 0 \longrightarrow \\
& \quad (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\
& \quad < \text{min} (vT-fd-sol-1 \\
& \quad \quad (\lambda n1. \text{real-of-int} \\
& \quad \quad \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v n1!(\text{Suc } 0)) 0]))) \\
& \quad \quad \quad (\lambda n1. \text{ if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& \quad \quad (\text{real-of-int} \\
& \quad \quad \quad (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0]))) + \\
& \quad \quad 1 \longrightarrow \\
& \quad (\neg \text{latch-rec-calc-output} \\
& \quad \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \\
& \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad x = \\
& \quad \quad 0 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& \quad \quad (\text{latch-rec-calc-output} \\
& \quad \quad \quad (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \\
& \quad \quad \quad \quad \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8 \\
& \quad \quad \quad \quad \text{then } 0 \text{ else } 1) \\
& \quad \quad (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \quad \quad \quad \text{else } 1) \\
& \quad \quad x = \\
& \quad \quad 0 \longrightarrow \\
& \quad \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& \quad \quad (\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\
& \quad \quad < \text{min} (vT-fd-sol-1 \\
& \quad \quad \quad (\lambda n1. \text{real-of-int}
\end{aligned}$$

$$\begin{aligned}
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v n1!(Suc 0)) 0 \rceil))) \\
& (\lambda n1. \text{if } hd (inouts_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - Suc 0)) \\
& (real-of-int \\
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil))) + \\
& 1 \longrightarrow \\
& length(inouts_v x) = 5 \wedge \\
& length(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. \text{if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge \\
& (\neg hd (inouts_v x) = 0 \longrightarrow \\
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& (\neg latch-rec-calc-output \\
& (\lambda n1. \text{if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \\
& \quad \text{then } 0 \text{ else } 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. \text{if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge \\
& (\neg int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge \\
& length(inouts_v' x) = 2 \wedge \\
& [0, 0] = inouts_v' x \wedge \\
& (latch-rec-calc-output \\
& (\lambda n1. \text{if } inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow \\
& \quad hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 \text{ then } 0 \\
& \quad \text{else } 1) \\
& x = \\
& 0 \longrightarrow \\
& length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)))) \wedge \\
& (\neg hd (inouts_v (x - Suc 0)) = 0 \longrightarrow \\
& (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)) < 0 \longrightarrow \\
& (hd (inouts_v x) = 0 \longrightarrow \\
& (real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil))) \\
& < min (vT-fd-sol-1
\end{aligned}$$

$(\lambda n1. \text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0)) \ 0])))$
 $(\lambda n1. \text{if hd} (\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) \ 0]))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\text{then } 0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge$
 $(\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0])))$
 $< \text{min} (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0)) \ 0])))$
 $(\lambda n1. \text{if hd} (\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int}$
 $(\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) \ 0]))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' \ x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' \ x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\text{hd} (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \ n1!2 = 8$

then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 else 1)
 x =
 0 \longrightarrow
 length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge
 (\neg latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8
 then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 then 0 else 1)
 (x - Suc 0) =
 0 \longrightarrow
 length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge
 (\neg hd (inouts_v x) = 0 \longrightarrow
 (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 \longrightarrow
 (\neg latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8
 then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 then 0 else 1)
 x =
 0 \wedge
 latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8
 then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 else 1)
 (x - Suc 0) =
 0 \longrightarrow
 length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge
 (latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8
 then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 else 1)
 x =
 0 \longrightarrow
 length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge
 (\neg latch-rec-calc-output
 ($\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v n1!2 = 8
 then 0 else 1)
 ($\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 then 0 else 1)
 (x - Suc 0) =
 0 \longrightarrow
 length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge
 (\neg int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 \longrightarrow
 length(inouts_v x) = 5 \wedge
 length(inouts_v' x) = 2 \wedge

$[0, 0] = \text{inouts}_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
(¬ latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)! (\text{Suc } 0)) 0]))) < 0 \longrightarrow$
 $(\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x! (\text{Suc } 0)) 0])))$
 $< \min (\text{vT-fd-sol-1}$
 $\quad (\lambda n1. \text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v n1! (\text{Suc } 0)) 0])))$
 $\quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $\quad (\text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)! (\text{Suc } 0)) 0]))) +$
 $\quad 1 \longrightarrow$
(¬ latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$
 $\quad \text{then } 0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow$
 $\quad \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v n1!2 = 8$
 $\quad \text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\quad \text{else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v x! (\text{Suc } 0)) 0])))$
 $< \min (\text{vT-fd-sol-1}$
 $\quad (\lambda n1. \text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v n1! (\text{Suc } 0)) 0])))$
 $\quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $\quad (\text{real-of-int}$
 $\quad \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)! (\text{Suc } 0)) 0]))) +$
 $\quad 1 \longrightarrow$

```

length(inouts_v x) = 5 ∧
length(inouts_v' x) = 2 ∧
[0, 0] = inouts_v' x ∧
(latch-rec-calc-output
  (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
    hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
    then 0 else 1)
  (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
    else 1)
  x =
  0 →
  length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x)) ∧
(¬ hd (inouts_v x) = 0 →
(int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
  (¬ latch-rec-calc-output
    (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
      hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
      then 0 else 1)
    (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0
      then 0 else 1)
    x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [1, 1] = inouts_v' x) ∧
  (latch-rec-calc-output
    (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
      hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
      then 0 else 1)
    (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
      else 1)
    x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x)) ∧
  (¬ int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
    length(inouts_v x) = 5 ∧
    length(inouts_v' x) = 2 ∧
    [0, 0] = inouts_v' x ∧
    (latch-rec-calc-output
      (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
        hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
        then 0 else 1)
      (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
        else 1)
      x =
      0 →
      length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x))))))

```

assume a2: $\forall x. ?P x$

assume a3: $hd (inouts_v' x) = 1$

assume a4: $\forall x. hd (tl (inouts_v' x)) = inouts_v x!(4)$

assume a5: $\forall xb \leq xa. inouts_v (Suc (x + xb))!(3) = 1 \wedge inouts_v (Suc (x + xb))!(2) = 8$

have len-inouts: $\forall x. length(inouts_v x) = 5$

using a2 **by** blast

have output-at-0: $inouts_v' 0 = [0, 0]$

using a2 **by** (smt One-nat-def zero-le-one)

have output-eq: $\forall x. hd (tl(inouts_v' x)) = hd(inouts_v' x)$

```

using a2 by (smt hd-Cons-tl list.inject not-gr0 tl-Nil)
have input-4-at-m: inouts_v x!(4) = 1
using a3 a4 output-eq by simp
have latch-at-m-1: latch-rec-calc-output
  (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
    hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
    then 0 else 1)
  (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0
    then 0 else 1)
  (Suc (x)) = 0
using input-4-at-m a5 by simp
have latch-m-1-to-p: ∀ q ≤ xa . latch-rec-calc-output
  (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
    hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
    then 0 else 1)
  (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0
    then 0 else 1)
  (Suc (x+q)) = 0
apply (rule allI)
proof -
  fix q::nat
  show q ≤ xa →
    latch-rec-calc-output
    (λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!(2)
= 8
      then 0 else 1)
    (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
else 1)
    (Suc (x + q)) = 0
  proof (induct q)
  case 0
  then show ?case
  using latch-at-m-1 by simp
  next
  case (Suc q)
  then show ?case
  apply (simp add: latch-rec-calc-output.elims)
  using a5 One-nat-def Suc-leD add-Suc-right diff-Suc-1 by smt
  qed
qed
have latch-at-p: latch-rec-calc-output
  (λn1. if inouts_v (n1 - Suc 0)!2 = 4 →
    hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v n1!2 = 8
    then 0 else 1)
  (λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0
    then 0 else 1)
  (Suc (x+xa)) = 0
  using latch-m-1-to-p by blast
  show inouts_v' (Suc (x + xa)) = inouts_v' 0
  using a2 latch-at-p by (smt output-at-0 zero-less-Suc)
qed

```

Secondly to verify the refinement relation for the feedback.

lemma req-02-ref: req-02-1-contract $f_D (4, 1) \sqsubseteq \text{plf-rise1shot-simp } f_D (4, 1)$
apply (rule feedback-mono[of 5 2])

using *SimBlock-req-02-1-contract* **apply** (*blast*)
using *post-landing-finalize-1-simblock* **apply** (*blast*)
using *req-02-ref-plf-rise1shot* **apply** (*blast*)
by (*auto*)

Thirdly to verify the requirement contract satisfied by the feedback of *req-02-1-contract*.

lemma *req-02-fd-ref*:

req-02-contract \sqsubseteq *req-02-1-contract* f_D (4, 1)

using *inps-req-02-1-contract* *outps-req-02-1-contract* **apply** (*simp add: PreFD-def PostFD-def*)

proof –

show *req-02-contract* \sqsubseteq ($\exists x \cdot$ (*true* \vdash_n

($\forall n \cdot \#_u(\$inouts(\langle n \rangle)_a) =_u \langle 4 \rangle \wedge \#_u(\$inouts'(\langle n \rangle)_a) =_u \langle 5 \rangle \wedge$
 $\$inouts'(\langle n \rangle)_a =_u \langle f-PreFD\ x\ 4 \rangle (\$inouts)_a(\langle n \rangle)_a$)) ; ;

req-02-1-contract ; ;

(*true* \vdash_n

($\forall n \cdot \#_u(\$inouts(\langle n \rangle)_a) =_u \langle 2 \rangle \wedge$
 $\#_u(\$inouts'(\langle n \rangle)_a) =_u \langle Suc\ 0 \rangle \wedge$
 $\$inouts'(\langle n \rangle)_a =_u \langle f-PostFD\ (Suc\ 0) \rangle (\$inouts)_a(\langle n \rangle)_a \wedge$
 $\langle uapply \rangle (\$inouts(\langle n \rangle)_a)_a(\langle Suc\ 0 \rangle)_a =_u \langle x\ n \rangle$))

apply (*simp (no-asm) add: req-02-1-contract-def req-02-contract-def*)

apply (*rel-simp*)

apply (*simp add: f-PostFD-def f-PreFD-def*)

proof –

fix *ok_v::bool and inouts_v::nat \Rightarrow real list and*

ok_v'::bool and inouts_v'::nat \Rightarrow real list and x::nat \Rightarrow real and

ok_v''::bool and inouts_v''::nat \Rightarrow real list and ok_v'''::bool and

inouts_v'''::nat \Rightarrow real list

assume *a1*: ($\forall xa. (hd\ (inouts_v\ xa \bullet [xa]) = 0 \vee hd\ (inouts_v\ xa \bullet [xa]) = 1) \wedge$

(*inouts_v* $xa \bullet [xa]!$ (*Suc* 0) = *c-door-open-time* \wedge

((*inouts_v* $xa \bullet [xa]!$ 3 = 0 \vee (*inouts_v* $xa \bullet [xa]!$ 3 = 1)) \longrightarrow

ok_v''' \wedge

($\forall x. length(inouts_v'''\ x) = 2) \wedge$

($\forall xa. hd\ (inouts_v'''\ xa) = 1 \wedge (\forall xa. hd\ (tl\ (inouts_v'''\ xa)) = (inouts_v\ xa \bullet [xa]!)4) \longrightarrow$

($\forall xb. (\forall xc \leq xb. (inouts_v\ (Suc\ (xa + xc)) \bullet [x\ (Suc\ (xa + xc))])!3 = 1 \wedge$

(*inouts_v* (Suc (xa + xc)) $\bullet [x\ (Suc\ (xa + xc))])!2 = 8) \longrightarrow$

inouts_v''' (Suc (xa + xb)) = [0, 0])

assume *a2*: *ok_v'''* \longrightarrow

ok_v' \wedge

($\forall xa. length(inouts_v'''\ xa) = 2 \wedge$

length(*inouts_v'* $xa) = Suc\ 0 \wedge$

inouts_v' $xa = take\ (Suc\ 0)\ (inouts_v'''\ xa) \bullet drop\ (Suc\ (Suc\ 0))\ (inouts_v'''\ xa) \wedge$

inouts_v'''\ xa!(*Suc* 0) = *x xa*)

assume *a3*: $\forall x. (hd\ (inouts_v\ x) = 0 \vee hd\ (inouts_v\ x) = 1) \wedge$

inouts_v $x!$ (*Suc* 0) = *c-door-open-time* \wedge (*inouts_v* $x!$ 3 = 0 \vee *inouts_v* $x!$ 3 = 1)

assume *a4*: $\forall xa. length(inouts_v\ xa) = 4 \wedge length(inouts_v''\ xa) = 5 \wedge$

inouts_v'' $xa = take\ 4\ (inouts_v\ xa) \bullet x\ xa \# drop\ 4\ (inouts_v\ xa)$

from *a4* **have** 1: $\forall xa. length(inouts_v\ xa) = 4$

by *blast*

have 2: ($\forall xa. (((hd\ (inouts_v\ xa \bullet [xa]) = 0 \vee hd\ (inouts_v\ xa \bullet [xa]) = 1) \wedge$

(*inouts_v* $xa \bullet [xa]!$ (*Suc* 0) = *c-door-open-time* \wedge

((*inouts_v* $xa \bullet [xa]!$ 3 = 0 \vee (*inouts_v* $xa \bullet [xa]!$ 3 = 1))

= ((*hd* (*inouts_v* $xa) = 0 \vee hd\ (inouts_v\ xa) = 1) \wedge$

inouts_v $xa!$ (*Suc* 0) = *c-door-open-time* \wedge (*inouts_v* $xa!$ 3 = 0 \vee *inouts_v* $xa!$ 3 = 1))))

using 1

by (metis Suc-mono Suc-numeral hd-append2 length-greater-0-conv nth-append numeral-2-eq-2
 numeral-3-eq-3 semiring-norm(2) semiring-norm(8) zero-less-Suc)
 have 3: ok_v'''
 using 2 a3 a1 by simp
 have 4: ok_v'
 using a2 3 by blast
 have 5: $\forall xa. inouts_v' xa = [hd (inouts_v''' xa)]$
 using 3 a2 by (metis append-eq-conv-conj length-Cons list.size(3) list-equal-size2 self-append-conv)
 have 6: $\forall xa. inouts_v''' xa!(Suc 0) = x xa$
 using a2 3 by blast
 have input-at-3: $\forall xa. (inouts_v xa \bullet [x xa])!3 = inouts_v xa!3$
 using 1 by (simp add: nth-append)
 have input-at-2: $\forall xa. (inouts_v xa \bullet [x xa])!2 = inouts_v xa!2$
 using 1 by (simp add: nth-append)
 have input-at-1: $\forall xa. (inouts_v xa \bullet [x xa])!1 = inouts_v xa!1$
 using 1 by (simp add: nth-append)
 have input-at-0: $\forall xa. (inouts_v xa \bullet [x xa])!0 = inouts_v xa!0$
 using 1 by (simp add: nth-append)
 have input-at-4: $\forall xa. (inouts_v xa \bullet [x xa])!4 = x xa$
 using 1 by (simp add: nth-append)
 have feedback: $(\forall xa. hd (tl(inouts_v''' xa)) = (inouts_v xa \bullet [x xa])!4) =$
 $(\forall xa. (inouts_v''' xa)!(Suc 0) = (x xa))$
 by (metis 3 One-nat-def a2 diff-Suc-1 hd-conv-nth input-at-4 length-greater-0-conv
 length-tl nth-tl numeral-2-eq-2 zero-less-one)
 have a1': $(\forall x. length(inouts_v''' x) = 2) \wedge$
 $(\forall xa. hd (inouts_v''' xa) = 1 \wedge (\forall xa. hd (tl (inouts_v''' xa)) = (inouts_v xa \bullet [x xa])!4) \longrightarrow$
 $(\forall xb. (\forall xc \leq xb. (inouts_v (Suc (xa + xc)) \bullet [x (Suc (xa + xc))])!3 = 1 \wedge$
 $(inouts_v (Suc (xa + xc)) \bullet [x (Suc (xa + xc))])!2 = 8) \longrightarrow$
 $inouts_v''' (Suc (xa + xb)) = [0, 0]))$
 using feedback a1 6 2 a3 input-at-3 input-at-2 by simp
 show $ok_v' \wedge$
 $(\forall x. length(inouts_v' x) = Suc 0) \wedge$
 $(\forall x. hd (inouts_v' x) = 1 \longrightarrow$
 $(\forall xa. (\forall xb \leq xa. inouts_v (Suc (x + xb))!3 = 1 \wedge inouts_v (Suc (x + xb))!2 = 8) \longrightarrow$
 $inouts_v' (Suc (x + xa)) = [0]))$
 apply (rule conjI)
 using 4 apply (simp)
 apply (rule conjI)
 using 3 a2 apply blast
 apply (rule allI, clarify)
 using a1' by (simp add: 3 5 a2 feedback input-at-2 input-at-3)

qed

qed

Finally, the requirement is held for the *post-landing-finalize-1* because of transitivity of refinement relation.

lemma req-02:

req-02-contract \sqsubseteq post-landing-finalize-1
 apply (simp only: post-landing-finalize-1-simp)
 using req-02-fd-ref req-02-ref by auto

C.5.3 Requirement 03

post-landing-finalize-req-03: The finalize event will not occur during flight.

During flight, *ac-on-ground* is false. According to Assumption 4 in the paper: "*door-closed*

must be true if *ac-on-ground* is false.", then *door-closed* is true during flight. Therefore, this requirement can be verified similarly as Requirement 04.

C.5.4 Requirement 04

post-landing-finalize-req-04: The finalize event will not be enabled while the aircraft door is closed.

Requirement 4: assumes

- *door-closed* and *ac-on-ground* are boolean,
- *door-open-time* is within $(0, \text{max-door-open-time})$

then it must guarantee that

- it has four inputs and one output,
- if the door is closed, then the output is always false (0).

abbreviation *req-04-contract* $\equiv ((\forall n::\text{nat} \cdot ($
 $\ll(\lambda x n. ($
 $(hd(x n) = 0 \vee hd(x n) = 1) \wedge (* \text{door-closed is boolean} *)$
 $((x n)!1 > 0 \wedge (x n)!1 < \text{max-door-open-time}) \wedge (* \text{door-open-time} *)$
 $((x n)!3 = 0 \vee (x n)!3 = 1) (* \text{ac-on-ground is boolean} *)$
 $))\gg$
 $(\&\text{inouts})_a (\ll n \gg)_a :: \text{sim-state upred})$
 \vdash_n
 $((\forall n::\text{nat} \cdot$
 $((\#_u(\$inouts (\ll n \gg)_a)) =_u \ll 4 \gg) \wedge$
 $((\#_u(\$inouts' (\ll n \gg)_a)) =_u \ll 1 \gg) \wedge$
 $((head_u(\$inouts (\ll n \gg)_a)) =_u 1) (* \text{door-closed is true} *)$
 $\Rightarrow (head_u(\$inouts' (\ll n \gg)_a) =_u 0)))$
 $))$

This is the contract for *post-landing-finalize-1* without the last feedback. Since *post-landing-finalize-1* is equal to *plf-rise1shot-simp* $f_D(4, 1)$, then this is the contract for *plf-rise1shot-simp*.

definition *req-04-1-contract* $\equiv ((\forall n::\text{nat} \cdot ($
 $\ll(\lambda x n. ($
 $(hd(x n) = 0 \vee hd(x n) = 1) \wedge (* \text{door-closed is boolean} *)$
 $((x n)!1 > 0 \wedge (x n)!1 < \text{max-door-open-time}) \wedge (* \text{door-open-time} *)$
 $((x n)!3 = 0 \vee (x n)!3 = 1) (* \text{ac-on-ground is boolean} *)$
 $))\gg$
 $(\&\text{inouts})_a (\ll n \gg)_a :: \text{sim-state upred})$
 \vdash_n
 $((\forall n::\text{nat} \cdot$
 $((\#_u(\$inouts (\ll n \gg)_a)) =_u \ll 5 \gg) \wedge$
 $((\#_u(\$inouts' (\ll n \gg)_a)) =_u \ll 2 \gg) \wedge$
 $((head_u(\$inouts (\ll n \gg)_a)) =_u 1) (* \text{door-closed is true} *)$
 $\Rightarrow (head_u(\$inouts' (\ll n \gg)_a) =_u 0) \wedge (head_u(\text{tail}_u(\$inouts' (\ll n \gg)_a)) =_u 0)))$
 $))$

lemma *SimBlock-req-04-1-contract*:
SimBlock 5 2 req-04-1-contract


```

apply (simp add: SimBlock-def req-04-1-contract-def)
apply (rel-auto)
apply (rule-tac x = λna. [0, 20, 4, 0, 0] in exI, simp)
by (rule-tac x = λna. [0, 0] in exI, simp)

```

```

lemma inps-req-04-1-contract:
  inps req-04-1-contract = 5
  using SimBlock-req-04-1-contract inps-P by blast

```

```

lemma outps-req-04-1-contract:
  outps req-04-1-contract = 2
  using SimBlock-req-04-1-contract outps-P by blast

```

In order to verify this requirement, firstly to verify the contract *req-04-1-contract* refined by *plf-rise1shot-simp*.

```

lemma req-04-ref-plf-rise1shot: req-04-1-contract ⊆ plf-rise1shot-simp
apply (simp add: FBlock-def plf-rise1shot-simp-def req-04-1-contract-def)
apply (rule ndesign-refine-intro)
apply simp
apply (unfold upred-defs urel-defs)
apply (simp add: fun-eq-iff relcomp-unfold OO-def
  lens-defs upred-defs alpha-splits Product-Type.split-beta)?
apply (transfer)
apply (simp; safe)
apply (rename-tac inoutsv inoutsv' x)
proof -
  fix inoutsv inoutsv'::nat ⇒ real list and x::nat
  assume a1: ∀ x. (hd (inoutsv x) = 0 ∨ hd (inoutsv x) = 1) ∧
    0 < inoutsv x!(Suc 0) ∧
    inoutsv x!(Suc 0) < max-door-open-time ∧
    (inoutsv x!3 = 0 ∨ inoutsv x!3 = 1)
  assume a2: ∀ x. (x ≤ Suc 0 →
    (hd (inoutsv 0) = 0 →
      (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!(Suc 0)) 0])) < 1 →
        (x = 0 → length(inoutsv 0) = 5 ∧ length(inoutsv' 0) = 2 ∧ [0, 0] = inoutsv' 0) ∧
        (0 < x →
          (hd (inoutsv x) = 0 →
            (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv x!(Suc 0)) 0])))
              < min 1 (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inoutsv 0!(Suc 0))
0]))) + 1 →
              (¬ latch-rec-calc-output
                (λn1. if inoutsv (n1 - Suc 0)!2 = 4 → hd (inoutsv n1) = 0 ∨ n1 = 0 ∨ ¬ inoutsv
n1!2 = 8
                  then 0 else 1)
                (λn1. if n1 = 0 ∨ ¬ inoutsv (n1 - Suc 0)!3 = 0 ∧ inoutsv (n1 - Suc 0)!4 = 0 then
0 else 1)
                  x =
                  0 →
                  length(inoutsv x) = 5 ∧ length(inoutsv' x) = 2 ∧ [1, 1] = inoutsv' x) ∧
                (latch-rec-calc-output
                  (λn1. if inoutsv (n1 - Suc 0)!2 = 4 → hd (inoutsv n1) = 0 ∨ n1 = 0 ∨ ¬ inoutsv
n1!2 = 8
                    then 0 else 1)
                  (λn1. if n1 = 0 ∨ ¬ inoutsv (n1 - Suc 0)!3 = 0 ∧ inoutsv (n1 - Suc 0)!4 = 0 then
0 else 1) x =

```

$$\begin{aligned}
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) \\
& \quad < \min 1 (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) \\
0 \rceil)))) + 1 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{hd} (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) < 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil))) < 1 \longrightarrow \\
& (x = 0 \longrightarrow \\
& \text{length}(\text{inouts}_v 0) = 5 \wedge \\
& \text{length}(\text{inouts}_v' 0) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' \\
0) \wedge \\
& (0 < x \longrightarrow \\
& (\text{hd} (\text{inouts}_v x) = 0 \longrightarrow
\end{aligned}$$

$(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))$
 $< \min 1 (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0))$
 $0 \rceil)))) + 1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) \ x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))$
 $< \min 1 (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ 0!(\text{Suc } 0))$
 $0 \rceil)))) + 1 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' \ x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' \ x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) \ x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge$
 $(\neg \text{hd } (\text{inouts}_v \ x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) \ x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v \ x) = 5 \wedge$

$length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)))) \wedge$
 $(\neg hd (inouts_v 0) = 0 \longrightarrow$
 $(int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!(Suc 0)) 0])) < 0 \longrightarrow$
 $(x = 0 \longrightarrow length(inouts_v 0) = 5 \wedge length(inouts_v' 0) = 2 \wedge [0, 0] = inouts_v' 0) \wedge$
 $(0 < x \longrightarrow$
 $(hd (inouts_v x) = 0 \longrightarrow$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min 0 (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!(Suc 0))$
 $0]))) + 1 \longrightarrow$
(\neg latch-rec-calc-output
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min 0 (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v 0!(Suc 0))$
 $0]))) + 1 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge$
 $(\neg hd (inouts_v x) = 0 \longrightarrow$
 $(int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 \longrightarrow$
(\neg latch-rec-calc-output
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
then 0 else 1)

$(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $(x = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v 0) = 5 \wedge$
 $\text{length}(\text{inouts}_v' 0) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v'$
 $0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))))$
 $< \min 0(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)))) + 1 \longrightarrow$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))))$
 $< \min 0(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil))))$

$0]])) + 1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))]])) \wedge$
 $(\neg x \leq \text{Suc } 0 \longrightarrow$
 $(\text{hd } (\text{inouts}_v (x - \text{Suc } 0)) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0])))$
 $< \text{min } (vT\text{-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0))$
 $0]])))))$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } (\text{Suc } 0)))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } (\text{Suc } 0))!(\text{Suc}$
 $0)) 0]])))] +$
 $1 \longrightarrow$
 $(\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$

$< \min (vT\text{-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \text{ n1!}(\text{Suc } 0))$
 $0 \rceil))))))$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v \text{ n1}) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0))! (\text{Suc } 0)$
 $0 \rceil)))))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \text{ n1}) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \text{ n1}) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \text{ n1}) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \text{ n1}) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x! (\text{Suc } 0)) 0 \rceil))))$
 $< \min (vT\text{-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \text{ n1!}(\text{Suc}$
 $0)) 0 \rceil))))))$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v \text{ n1}) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0))! (\text{Suc}$
 $0)) 0 \rceil)))))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \text{ n1}) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$

$$\begin{aligned} & \text{then } 0 \text{ else } 1) \\ \text{else } 1) \ x = & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\ n1!2 = 8 & \\ & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\ 0 \text{ else } 1) & \\ & (x - \text{Suc } 0) = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\ & (\neg \text{hd}(\text{inouts}_v x) = 0 \longrightarrow \\ & (\text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\ & (\neg \text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\ n1!2 = 8 & \\ & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\ 0 \text{ else } 1) & \\ & x = \\ & 0 \wedge \\ & \text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\ n1!2 = 8 & \\ & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ \text{else } 1) & \\ & (x - \text{Suc } 0) = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\ & (\text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\ n1!2 = 8 & \\ & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\ \text{else } 1) \ x = & \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\ & (\neg \text{latch-rec-calc-output} \\ & (\lambda n1. \text{ if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\ n1!2 = 8 & \\ & \text{then } 0 \text{ else } 1) \\ & (\lambda n1. \text{ if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\ 0 \text{ else } 1) & \\ & (x - \text{Suc } 0) = \\ & 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\ & (\neg \text{int32}(\text{RoundZero}(\text{real-of-int}[\text{Rate} * \text{max}(\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow \\ & \text{length}(\text{inouts}_v x) = 5 \wedge \\ & \text{length}(\text{inouts}_v' x) = 2 \wedge \\ & [0, 0] = \text{inouts}_v' x \wedge \\ & (\text{latch-rec-calc-output} \end{aligned}$$

$n1!2 = 8$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0])))$
 $< \text{min } (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0))$
 $0])))$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } (\text{Suc } 0)))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } (\text{Suc } 0))!(\text{Suc}$
 $0)) 0]))) +$
 $1 \longrightarrow$
 $(\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc } 0))$
 $0])))$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0]))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc}$
 $0)) 0])))$

$$\begin{aligned}
& (\lambda n1. \text{if } \text{hd}(\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ (x - \text{Suc } 0))! (\text{Suc } 0)) \\
& 0)) \ 0 \rceil)))) + \\
& \quad 1 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v \ x) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' \ x) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' \ x \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \text{else } 1) \ x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)) \wedge \\
& (\neg \text{hd}(\text{inouts}_v \ x) = 0 \longrightarrow \\
& (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ x)! (\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& 0 \text{ else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \text{else } 1) \ x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)) \wedge \\
& (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ x)! (\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' \ x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' \ x \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \text{else } 1) \ x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x)))) \wedge \\
& (\neg \text{hd}(\text{inouts}_v \ (x - \text{Suc } 0)) = 0 \longrightarrow \\
& (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ (x - \text{Suc } 0))! (\text{Suc } 0)) \ 0 \rceil)) < 0 \longrightarrow \\
& (\text{hd}(\text{inouts}_v \ x) = 0 \longrightarrow \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ x)! (\text{Suc } 0)) \ 0 \rceil))) \\
& < \min(\text{vT-fd-sol-1} \\
& \quad (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ n1)! (\text{Suc } 0)) \\
& 0 \rceil)))))) \\
& \quad (\lambda n1. \text{if } \text{hd}(\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ (x - \text{Suc } 0))! (\text{Suc } 0))
\end{aligned}$$

$0])])]) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } (vT\text{-fd-sol-1}$
 $(\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc}$
 $0)) 0])))$
 $0)) 0])])]) +$
 $(\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0))!(\text{Suc}$
 $0)) 0]))) +$
 $1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$

$(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
 $\text{else } 1) x =$

$0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $(x - Suc 0) =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x))) \wedge$
 $(\neg int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0])) < 0 \longrightarrow$
 $(hd (inouts_v x) = 0 \longrightarrow$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min (vT-fd-sol-1$
 $(\lambda n1. real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc 0))$
 $0])))$
 $(\lambda n1. if hd (inouts_v n1) = 0 then 1 else 0) (x - Suc 0))$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc 0))$
 $0]))) +$
 $1 \longrightarrow$
 $(\neg latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge$
 $(\neg real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min (vT-fd-sol-1$
 $(\lambda n1. real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc$
 $0)) 0])))$
 $(\lambda n1. if hd (inouts_v n1) = 0 then 1 else 0) (x - Suc 0))$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc$
 $0)) 0]))) +$
 $1 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$

```

    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x)) ∧
(¬ hd (inouts_v x) = 0 →
(int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
(¬ latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then
0 else 1)
    x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [1, 1] = inouts_v' x) ∧
(latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
else 1) x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x)) ∧
(¬ int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
length(inouts_v x) = 5 ∧
length(inouts_v' x) = 2 ∧
[0, 0] = inouts_v' x ∧
(latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
else 1) x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x))))))
assume a3: hd (inouts_v x) = 1
have 1: ∀ x. (inouts_v x!(Suc 0)) > 0 ∧ (inouts_v x!(Suc 0)) < max-door-open-time
using a1 by blast
have 2: ∀ x. int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) ≥ 0 ∧
int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < (Rate * max-door-open-time
+ 1)
apply (rule allI)
proof -
fix xx::nat
have 0: Rate * max (inouts_v xx!(Suc 0)) 0 < Rate * max-door-open-time ∧ Rate * max x 0 ≥ 0
using 1 by simp
have 1: [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max (inouts_v xx!(Suc 0)) 0 + 1)
using ceiling-correct by linarith
then have [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1)
using 0 1 by linarith
then have 2: [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1) ∧
[Rate * max (inouts_v xx!(Suc 0)) 0] ≥ 0
using 0 by (smt ceiling-le-zero ceiling-zero)
have 3: real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1) ∧
real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0] ≥ 0
using 2 by (simp)
have 4: RoundZero (real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0])

```

$$= \lfloor \text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil \rfloor$$
using *RoundZero-def* **by** (*simp*)

have 5: $\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil) < (\text{Rate} * \text{max-door-open-time} + 1) \wedge$

$$\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil) \geq 0$$
using 3 4 **by** *auto*

have 51: $\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil) < (\text{Rate} * 214748364 + 1) \wedge$

$$\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil) \geq 0$$
using 5 1 **by** *auto*

have 6: $\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)) = \text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)$
using 51 *int32-eq 1* **by** *simp*

have 7: $\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)) < (\text{Rate} * \text{max-door-open-time} + 1) \wedge$

$$\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)) \geq 0$$
using 5 6 **by** (*simp*)

show $0 \leq \text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)) \wedge$

$$\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil)) < \text{Rate} * \text{max-door-open-time} + 1$$
using 7 **by** *blast*

qed

show $\text{hd}(\text{inouts}_v' x) = 0$
using 2 a2 a3 a1 *neq0-conv list.sel(1)* **by** (*smt*)

next

fix $\text{inouts}_v \text{ inouts}_v' :: \text{nat} \Rightarrow \text{real list}$ **and** $x :: \text{nat}$

assume a1: $\forall x. (\text{hd}(\text{inouts}_v x) = 0 \vee \text{hd}(\text{inouts}_v x) = 1) \wedge$

$$0 < \text{inouts}_v \text{ x!}(\text{Suc } 0) \wedge$$

$$\text{inouts}_v \text{ x!}(\text{Suc } 0) < \text{max-door-open-time} \wedge$$

$$(\text{inouts}_v \text{ x!}3 = 0 \vee \text{inouts}_v \text{ x!}3 = 1)$$

assume a2: $\forall x. (x \leq \text{Suc } 0 \longrightarrow$

$$(\text{hd}(\text{inouts}_v 0) = 0 \longrightarrow$$

$$(\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)) < 1 \longrightarrow$$

$$(x = 0 \longrightarrow \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' 0) \wedge$$

$$(0 < x \longrightarrow$$

$$(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$$

$$(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \text{ x!}(\text{Suc } 0)) \ 0 \rceil))))$$

$$< \min 1 (\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int } \lceil \text{Rate} * \max(\text{inouts}_v \ 0!(\text{Suc } 0)) \ 0 \rceil)))) + 1 \longrightarrow$$

$$(\neg \text{latch-rec-calc-output}$$

$$(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$$

$$n1!2 = 8$$

$$\text{then } 0 \text{ else } 1)$$

$$(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$$

$$0 \text{ else } 1)$$

$$x =$$

$$0 \longrightarrow$$

$$\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$$

$$(\text{latch-rec-calc-output}$$

$$(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$$

$$n1!2 = 8$$

$$\text{then } 0 \text{ else } 1)$$

$$(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$$

$$0 \text{ else } 1) x =$$

$$0 \longrightarrow$$

$$\begin{aligned}
& \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) \\
& \quad < \min 1 (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) \\
0 \rceil)))) + 1 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' x \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{hd} (\text{inouts}_v x) = 0 \longrightarrow \\
& (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) < 0 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))) < 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' x \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 \\
& \quad \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil))) < 1 \longrightarrow \\
& \quad (x = 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v 0) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' 0) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' \\
0) \wedge \\
& \quad (0 < x \longrightarrow \\
& \quad (\text{hd} (\text{inouts}_v x) = 0 \longrightarrow \\
& \quad (\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))))
\end{aligned}$$

$$\begin{aligned}
& < \min 1 \text{ (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts}_v \text{ 0!(Suc 0))} \\
& 0]))) + 1 \longrightarrow \\
& \quad (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if inouts}_v \text{ (n1 - Suc 0)!2 = 4} \longrightarrow \text{hd (inouts}_v \text{ n1) = 0} \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then 0 else 1}) \\
& \quad (\lambda n1. \text{if n1 = 0} \vee \neg \text{inouts}_v \text{ (n1 - Suc 0)!3 = 0} \wedge \text{inouts}_v \text{ (n1 - Suc 0)!4 = 0 then} \\
& 0 \text{ else 1}) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \text{length(inouts}_v' \text{ x) = 2} \wedge [1, 1] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if inouts}_v \text{ (n1 - Suc 0)!2 = 4} \longrightarrow \text{hd (inouts}_v \text{ n1) = 0} \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then 0 else 1}) \\
& \quad (\lambda n1. \text{if n1 = 0} \vee \neg \text{inouts}_v \text{ (n1 - Suc 0)!3 = 0} \wedge \text{inouts}_v \text{ (n1 - Suc 0)!4 = 0 then} \\
& 0 \text{ else 1}) \text{ x =} \\
& \quad 0 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \text{length(inouts}_v' \text{ x) = 2} \wedge [0, 0] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\neg \text{real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts}_v \text{ x!(Suc 0)) 0])))} \\
& \quad < \min 1 \text{ (real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts}_v \text{ 0!(Suc 0))} \\
& 0]))) + 1 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \\
& \quad \text{length(inouts}_v' \text{ x) = 2} \wedge \\
& \quad [0, 0] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if inouts}_v \text{ (n1 - Suc 0)!2 = 4} \longrightarrow \text{hd (inouts}_v \text{ n1) = 0} \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then 0 else 1}) \\
& \quad (\lambda n1. \text{if n1 = 0} \vee \neg \text{inouts}_v \text{ (n1 - Suc 0)!3 = 0} \wedge \text{inouts}_v \text{ (n1 - Suc 0)!4 = 0 then} \\
& 0 \text{ else 1}) \text{ x =} \\
& \quad 0 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \text{length(inouts}_v' \text{ x) = 2} \wedge [0, 0] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\neg \text{hd (inouts}_v \text{ x) = 0} \longrightarrow \\
& \quad (\text{int32 (RoundZero (real-of-int [Rate * max (inouts}_v \text{ x!(Suc 0)) 0])) < 0} \longrightarrow \\
& \quad (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if inouts}_v \text{ (n1 - Suc 0)!2 = 4} \longrightarrow \text{hd (inouts}_v \text{ n1) = 0} \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then 0 else 1}) \\
& \quad (\lambda n1. \text{if n1 = 0} \vee \neg \text{inouts}_v \text{ (n1 - Suc 0)!3 = 0} \wedge \text{inouts}_v \text{ (n1 - Suc 0)!4 = 0 then} \\
& 0 \text{ else 1}) \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \text{length(inouts}_v' \text{ x) = 2} \wedge [1, 1] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if inouts}_v \text{ (n1 - Suc 0)!2 = 4} \longrightarrow \text{hd (inouts}_v \text{ n1) = 0} \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \quad \text{then 0 else 1}) \\
& \quad (\lambda n1. \text{if n1 = 0} \vee \neg \text{inouts}_v \text{ (n1 - Suc 0)!3 = 0} \wedge \text{inouts}_v \text{ (n1 - Suc 0)!4 = 0 then} \\
& 0 \text{ else 1}) \text{ x =} \\
& \quad 0 \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \text{length(inouts}_v' \text{ x) = 2} \wedge [0, 0] = \text{inouts}_v' \text{ x} \wedge \\
& \quad (\neg \text{int32 (RoundZero (real-of-int [Rate * max (inouts}_v \text{ x!(Suc 0)) 0])) < 0} \longrightarrow \\
& \quad \text{length(inouts}_v \text{ x) = 5} \wedge \\
& \quad \text{length(inouts}_v' \text{ x) = 2} \wedge
\end{aligned}$$

$[0, 0] = \text{inouts}_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v 0) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(x = 0 \longrightarrow \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v' 0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } 0 (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0))$
 $0]))) + 1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min } 0 (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v 0!(\text{Suc } 0))$
 $0]))) + 1 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1) x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$

0 else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ then
 0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ then
 0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow$
 $(x = 0 \longrightarrow$
 $\text{length}(\text{inouts}_v 0) = 5 \wedge$
 $\text{length}(\text{inouts}_v' 0) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' 0 \wedge \text{length}(\text{inouts}_v 0) = 5 \wedge \text{length}(\text{inouts}_v' 0) = 2 \wedge [0, 0] = \text{inouts}_v'$
 $0) \wedge$
 $(0 < x \longrightarrow$
 $(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))))$
 $< \min 0(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)))) + 1 \longrightarrow$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ then
 0 else 1)
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
(latch-rec-calc-output
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0$ then
 0 else 1) $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil))))$
 $< \min 0(\text{real-of-int}(\text{int32}(\text{RoundZero}(\text{real-of-int} \lceil \text{Rate} * \max(\text{inouts}_v 0!(\text{Suc } 0)) 0 \rceil)))) + 1 \longrightarrow$

$$\begin{aligned}
& \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' x \wedge \\
& (\text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \\
& \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
0 \text{ else } 1) x = & \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& \quad (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow \\
& \quad (\neg \text{latch-rec-calc-output} \\
n1!2 = 8 & \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& \quad \text{then } 0 \text{ else } 1) \\
0 \text{ else } 1) & \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then} \\
& \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& (\text{latch-rec-calc-output} \\
n1!2 = 8 & \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& \quad \text{then } 0 \text{ else } 1) \\
0 \text{ else } 1) x = & \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge \\
& (\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) < 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' x \wedge \\
& \quad (\text{latch-rec-calc-output} \\
n1!2 = 8 & \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& \quad \text{then } 0 \text{ else } 1) \\
0 \text{ else } 1) x = & \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)))))) \wedge \\
& (\neg x \leq \text{Suc } 0 \longrightarrow \\
& \quad (\text{hd } (\text{inouts}_v (x - \text{Suc } 0)) = 0 \longrightarrow \\
& \quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0)) 0 \rceil)) \\
& \quad < \min (\text{vT-fd-sol-1} \\
& \quad \quad (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v n1!(\text{Suc } 0)) \\
0 \rceil)))))) \\
& \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } (\text{Suc } 0))) \\
& \quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v (x - \text{Suc } (\text{Suc } 0))!(\text{Suc} \\
0 \rceil)))))) + \\
& \quad 1 \longrightarrow \\
& \quad (\text{hd } (\text{inouts}_v x) = 0 \longrightarrow \\
& \quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v x!(\text{Suc } 0)) 0 \rceil)) \\
& \quad < \min (\text{vT-fd-sol-1}
\end{aligned}$$

$$\begin{aligned}
& (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc } 0)) \\
& 0 \rceil)))) \\
& (\lambda n1. \text{if } \text{hd } (\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ (x - \text{Suc } 0))!(\text{Suc } 0)) \\
& 0 \rceil)))) + \\
& 1 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } \\
& 0 \text{ else } 1) \\
& x = \\
& 0 \wedge \\
& \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \text{else } 1) \\
& (x - \text{Suc } 0) = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [1, 1] = \text{inouts}_v' \ x) \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
& \text{else } 1) \ x = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \text{then } 0 \text{ else } 1) \\
& (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v \ (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v \ (n1 - \text{Suc } 0)!4 = 0 \text{ then } \\
& 0 \text{ else } 1) \\
& (x - \text{Suc } 0) = \\
& 0 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \text{length}(\text{inouts}_v' \ x) = 2 \wedge [0, 0] = \text{inouts}_v' \ x) \wedge \\
& (\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ x!(\text{Suc } 0)) \ 0 \rceil)))) \\
& < \text{min } (vT\text{-fd-sol-1} \\
& (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ n1!(\text{Suc} \\
& 0)) \ 0 \rceil)))) \\
& (\lambda n1. \text{if } \text{hd } (\text{inouts}_v \ n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } \lceil \text{Rate} * \max (\text{inouts}_v \ (x - \text{Suc } 0))!(\text{Suc} \\
& 0)) \ 0 \rceil)))) + \\
& 1 \longrightarrow \\
& \text{length}(\text{inouts}_v \ x) = 5 \wedge \\
& \text{length}(\text{inouts}_v' \ x) = 2 \wedge \\
& [0, 0] = \text{inouts}_v' \ x \wedge \\
& (\text{latch-rec-calc-output} \\
& (\lambda n1. \text{if } \text{inouts}_v \ (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v \ n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
& n1!2 = 8 \\
& \text{then } 0 \text{ else } 1)
\end{aligned}$$

$(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1) x =
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1)
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{hd} (\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1)
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1)
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1) x =
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
n1!2 = 8
 $\text{then } 0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
0 else 1)
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x)) \wedge$
 $(\neg \text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0])) < 0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge$
 $\text{length}(\text{inouts}_v' x) = 2 \wedge$
 $[0, 0] = \text{inouts}_v' x \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd} (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$

$n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1) x =
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $(x - \text{Suc } 0) =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x))) \wedge$
 $(\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0])))$
 $< \text{min} (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v n1!(\text{Suc } 0))$
 $0])))$
 $(\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc} (\text{Suc } 0)))$
 $(\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc} (\text{Suc } 0))!(\text{Suc}$
 $0)) 0]))) +$
 $1 \longrightarrow$
 $(\text{hd}(\text{inouts}_v x) = 0 \longrightarrow$
 $(\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min} (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v n1!(\text{Suc } 0))$
 $0])))$
 $(\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$
 $(\text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc } 0))$
 $0]))) +$
 $1 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then}$
 $0 \text{ else } 1)$
 $x =$
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge$
 $(\text{latch-rec-calc-output}$
 $(\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd}(\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v$
 $n1!2 = 8$
then 0 else 1)
 $(\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0$
else 1) x =
 $0 \longrightarrow$
 $\text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge$
 $(\neg \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v x!(\text{Suc } 0)) 0])))$
 $< \text{min} (\text{vT-fd-sol-1}$
 $(\lambda n1. \text{real-of-int} (\text{int32} (\text{RoundZero} (\text{real-of-int} [\text{Rate} * \text{max} (\text{inouts}_v n1!(\text{Suc}$
 $0)) 0])))$
 $(\lambda n1. \text{if } \text{hd}(\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0))$

$(real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)))) +$
 $1 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge$
 $(\neg hd (inouts_v x) = 0 \longrightarrow$
 $(int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow$
 $(\neg latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)) \wedge$
 $(\neg int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)) < 0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x)))) \wedge$
 $(\neg hd (inouts_v (x - Suc 0)) = 0 \longrightarrow$
 $(int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0 \rceil)) < 0 \longrightarrow$
 $(hd (inouts_v x) = 0 \longrightarrow$
 $(real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v x!(Suc 0)) 0 \rceil)))$
 $< min (vT-fd-sol-1$
 $(\lambda n1. real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v n1!(Suc 0))$
 $0 \rceil))))$
 $0 \rceil)))) +$
 $(\lambda n1. if hd (inouts_v n1) = 0 then 1 else 0) (x - Suc 0))$
 $(real-of-int (int32 (RoundZero (real-of-int \lceil Rate * max (inouts_v (x - Suc 0)!(Suc 0))$
 $0 \rceil)))) +$

$$\begin{aligned}
& 1 \longrightarrow \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
0 \text{ else } 1) & \quad x = \\
& \quad 0 \wedge \\
& \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
\text{else } 1) & \quad (x - \text{Suc } 0) = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [1, 1] = \text{inouts}_v' x) \wedge \\
& \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
\text{else } 1) & \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } \\
0 \text{ else } 1) & \quad (x - \text{Suc } 0) = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v x!(\text{Suc } 0)) 0]))) \\
& \quad < \text{min } (vT\text{-fd-sol-1} \\
& \quad \quad (\lambda n1. \text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v n1!(\text{Suc} \\
0)) 0]))) \\
& \quad \quad (\lambda n1. \text{if } \text{hd } (\text{inouts}_v n1) = 0 \text{ then } 1 \text{ else } 0) (x - \text{Suc } 0)) \\
& \quad \quad (\text{real-of-int } (\text{int32 } (\text{RoundZero } (\text{real-of-int } [\text{Rate} * \text{max } (\text{inouts}_v (x - \text{Suc } 0)!(\text{Suc} \\
0)) 0]))) + \\
& \quad 1 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \\
& \quad \text{length}(\text{inouts}_v' x) = 2 \wedge \\
& \quad [0, 0] = \text{inouts}_v' x \wedge \\
& \text{latch-rec-calc-output} \\
& \quad (\lambda n1. \text{if } \text{inouts}_v (n1 - \text{Suc } 0)!2 = 4 \longrightarrow \text{hd } (\text{inouts}_v n1) = 0 \vee n1 = 0 \vee \neg \text{inouts}_v \\
n1!2 = 8 & \quad \text{then } 0 \text{ else } 1) \\
& \quad (\lambda n1. \text{if } n1 = 0 \vee \neg \text{inouts}_v (n1 - \text{Suc } 0)!3 = 0 \wedge \text{inouts}_v (n1 - \text{Suc } 0)!4 = 0 \text{ then } 0 \\
\text{else } 1) & \quad x = \\
& \quad 0 \longrightarrow \\
& \quad \text{length}(\text{inouts}_v x) = 5 \wedge \text{length}(\text{inouts}_v' x) = 2 \wedge [0, 0] = \text{inouts}_v' x) \wedge \\
& (\neg \text{latch-rec-calc-output}
\end{aligned}$$

$n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then}$
 $(x - Suc\ 0) =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [0, 0] = inouts_v'\ x)) \wedge$
 $(\neg hd (inouts_v\ x) = 0 \longrightarrow$
 $(int32 (RoundZero (real-of-int [Rate * max (inouts_v\ x!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $(\neg \text{latch-rec-calc-output}$
 $n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $0 \text{ else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then}$
 $x =$
 $0 \wedge$
 $\text{latch-rec-calc-output}$
 $n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $\text{else } 1)$
 $(\lambda n1. \text{if } n1 = 0 \vee \neg inouts_v (n1 - Suc\ 0)!3 = 0 \wedge inouts_v (n1 - Suc\ 0)!4 = 0 \text{ then } 0$
 $(x - Suc\ 0) =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [1, 1] = inouts_v'\ x) \wedge$
 $(\text{latch-rec-calc-output}$
 $n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $\text{else } 1) x =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [0, 0] = inouts_v'\ x) \wedge$
 $(\neg \text{latch-rec-calc-output}$
 $n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $0 \text{ else } 1)$
 $(x - Suc\ 0) =$
 $0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge length(inouts_v'\ x) = 2 \wedge [0, 0] = inouts_v'\ x)) \wedge$
 $(\neg int32 (RoundZero (real-of-int [Rate * max (inouts_v\ x!(Suc\ 0))\ 0])) < 0 \longrightarrow$
 $length(inouts_v\ x) = 5 \wedge$
 $length(inouts_v'\ x) = 2 \wedge$
 $[0, 0] = inouts_v'\ x \wedge$
 $(\text{latch-rec-calc-output}$
 $n1!2 = 8$
 $(\lambda n1. \text{if } inouts_v (n1 - Suc\ 0)!2 = 4 \longrightarrow hd (inouts_v\ n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $\text{then } 0 \text{ else } 1)$
 $\text{else } 1) x =$
 $0 \longrightarrow$

$length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $(x - Suc 0) =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x))) \wedge$
 $(\neg int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc 0)) 0])) < 0 \longrightarrow$
 $(hd (inouts_v x) = 0 \longrightarrow$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min (vT-fd-sol-1$
 $(\lambda n1. real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc 0))$
 $0])))$
 $(\lambda n1. if hd (inouts_v n1) = 0 then 1 else 0) (x - Suc 0))$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc 0))$
 $0]))) +$
 $1 \longrightarrow$
 $(\neg latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then$
 $0 else 1)$
 $x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [1, 1] = inouts_v' x) \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$
 $length(inouts_v x) = 5 \wedge length(inouts_v' x) = 2 \wedge [0, 0] = inouts_v' x) \wedge$
 $(\neg real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])))$
 $< min (vT-fd-sol-1$
 $(\lambda n1. real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v n1!(Suc$
 $0)) 0])))$
 $(\lambda n1. if hd (inouts_v n1) = 0 then 1 else 0) (x - Suc 0))$
 $(real-of-int (int32 (RoundZero (real-of-int [Rate * max (inouts_v (x - Suc 0)!(Suc$
 $0)) 0]))) +$
 $1 \longrightarrow$
 $length(inouts_v x) = 5 \wedge$
 $length(inouts_v' x) = 2 \wedge$
 $[0, 0] = inouts_v' x \wedge$
 $(latch-rec-calc-output$
 $(\lambda n1. if inouts_v (n1 - Suc 0)!2 = 4 \longrightarrow hd (inouts_v n1) = 0 \vee n1 = 0 \vee \neg inouts_v$
 $n1!2 = 8$
 $then 0 else 1)$
 $(\lambda n1. if n1 = 0 \vee \neg inouts_v (n1 - Suc 0)!3 = 0 \wedge inouts_v (n1 - Suc 0)!4 = 0 then 0$
 $else 1) x =$
 $0 \longrightarrow$

```

    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x))) ∧
(¬ hd (inouts_v x) = 0 →
(int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
(¬ latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then
0 else 1)
    x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [1, 1] = inouts_v' x) ∧
(latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
else 1) x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x) ∧
(¬ int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < 0 →
length(inouts_v x) = 5 ∧
length(inouts_v' x) = 2 ∧
[0, 0] = inouts_v' x ∧
(latch-rec-calc-output
(λn1. if inouts_v (n1 - Suc 0)!2 = 4 → hd (inouts_v n1) = 0 ∨ n1 = 0 ∨ ¬ inouts_v
n1!2 = 8
    then 0 else 1)
(λn1. if n1 = 0 ∨ ¬ inouts_v (n1 - Suc 0)!3 = 0 ∧ inouts_v (n1 - Suc 0)!4 = 0 then 0
else 1) x =
    0 →
    length(inouts_v x) = 5 ∧ length(inouts_v' x) = 2 ∧ [0, 0] = inouts_v' x))))))
assume a3: hd (inouts_v x) = 1
have 1: ∀ x. (inouts_v x!(Suc 0)) > 0 ∧ (inouts_v x!(Suc 0)) < max-door-open-time
    using a1 by blast
have 2: ∀ x. int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) ≥ 0 ∧
    int32 (RoundZero (real-of-int [Rate * max (inouts_v x!(Suc 0)) 0])) < (Rate * max-door-open-time
+ 1)
apply (rule allI)
proof -
    fix xx::nat
have 0: Rate * max (inouts_v xx!(Suc 0)) 0 < Rate * max-door-open-time ∧ Rate * max x 0 ≥ 0
    using 1 by simp
have 1: [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max (inouts_v xx!(Suc 0)) 0 + 1)
    using ceiling-correct by linarith
then have [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1)
    using 0 1 by linarith
then have 2: [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1) ∧
    [Rate * max (inouts_v xx!(Suc 0)) 0] ≥ 0
    using 0 by (smt ceiling-le-zero ceiling-zero)
have 3: real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0] < (Rate * max-door-open-time + 1) ∧
    real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0] ≥ 0
    using 2 by (simp)
have 4: RoundZero (real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0])
    = [real-of-int [Rate * max (inouts_v xx!(Suc 0)) 0]]

```

```

using RoundZero-def by (simp)
have 5: RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ) < (Rate * max-door-open-time
+ 1) ∧
      RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ) ≥ 0
using 3 4 by auto
have 51: RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ) < (Rate * 214748364 +
1) ∧
      RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ) ≥ 0
using 5 1 by auto
have 6: int32 (RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ))
  = RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ )
using 51 int32-eq 1 by simp
have 7: int32 (RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ ))
  < (Rate * max-door-open-time + 1) ∧
      int32 (RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ )) ≥ 0
using 5 6 by (simp)
show 0 ≤ int32 (RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ )) ∧
int32 (RoundZero (real-of-int  $\lceil$ Rate * max (inoutsv xx!(Suc 0)) 0 $\rceil$ )) < Rate * max-door-open-time
+ 1
using 7 by blast
qed
show hd (tl (inoutsv' x)) = 0
using 2 a2 a3 a1 neq0-conv list.sel(1) list.sel(3) by (smt)
qed

```

Secondly to verify the refinement relation for the feedback.

```

lemma req-04-ref: req-04-1-contract fD (4, 1) ⊆ plf-rise1shot-simp fD (4, 1)
apply (rule feedback-mono[of 5 2])
using SimBlock-req-04-1-contract apply (blast)
using post-landing-finalize-1-simblock apply (blast)
using req-04-ref-plf-rise1shot apply (blast)
by (auto)

```

Thirdly to verify the requirement contract satisfied by the feedback of req-04-1-contract.

```

lemma req-04-fd-ref:
  req-04-contract ⊆ req-04-1-contract fD (4, 1)
using inps-req-04-1-contract outps-req-04-1-contract apply (simp add: PreFD-def PostFD-def)
proof –
  show (∀ n • «λx n. (hd (x n) = 0 ∨ hd (x n) = 1) ∧
      0 < x n!(Suc 0) ∧
      x n!(Suc 0) < max-door-open-time ∧
      (x n!3 = 0 ∨ x n!3 = 1)»(&inouts)a(«n»)a) ⊢n
    (∀ n • #u($inouts(«n»)a) =u «4» ∧
      #u($inouts'(«n»)a) =u «Suc 0» ∧ (headu($inouts(«n»)a) =u 1 ⇒ headu($inouts'(«n»)a)
=u 0))
  ⊆
    (∃ x • (true ⊢n
      (∀ n • #u($inouts(«n»)a) =u «4» ∧
        #u($inouts'(«n»)a) =u «5» ∧ $inouts'(«n»)a =u «f-PreFD x 4»($inouts)a(«n»)a))
  ;;
  req-04-1-contract ;;
  (true ⊢n
    (∀ n • #u($inouts(«n»)a) =u «2» ∧
      #u($inouts'(«n»)a) =u «Suc 0» ∧
      $inouts'(«n»)a =u «f-PostFD (Suc 0)»($inouts)a(«n»)a ∧

```

$\llbracket \text{uapply} \rrbracket (\$ \text{inouts} (\llbracket n \rrbracket)_a)_a (\llbracket \text{Suc } 0 \rrbracket)_a =_u \llbracket x \ n \rrbracket \rrbracket \rrbracket$

apply (*simp* (*no-asm*) *add: req-04-1-contract-def*)

apply (*rel-simp*)

apply (*simp* *add: f-PostFD-def f-PreFD-def*)

proof –

fix $ok_v :: \text{bool}$ **and** $inouts_v :: \text{nat} \Rightarrow \text{real list}$ **and**
 $ok_v' :: \text{bool}$ **and** $inouts_v' :: \text{nat} \Rightarrow \text{real list}$ **and** $x :: \text{nat} \Rightarrow \text{real}$ **and**
 $ok_v'' :: \text{bool}$ **and** $inouts_v'' :: \text{nat} \Rightarrow \text{real list}$ **and** $ok_v''' :: \text{bool}$ **and**
 $inouts_v''' :: \text{nat} \Rightarrow \text{real list}$

assume $a1: (\forall xa. (\text{hd } (inouts_v \ x) \bullet [x \ xa]) = 0 \vee \text{hd } (inouts_v \ x) \bullet [x \ xa]) = 1) \wedge$
 $0 < (inouts_v \ x) \bullet [x \ xa]!(\text{Suc } 0) \wedge$
 $(inouts_v \ x) \bullet [x \ xa]!(\text{Suc } 0) < \text{max-door-open-time} \wedge$
 $((inouts_v \ x) \bullet [x \ xa]!\exists = 0 \vee (inouts_v \ x) \bullet [x \ xa]!\exists = 1)) \longrightarrow$
 $ok_v''' \wedge$
 $(\forall xa. \text{length}(inouts_v''' \ x) = 2 \wedge$
 $(\text{hd } (inouts_v \ x) \bullet [x \ xa]) = 1 \longrightarrow$
 $\text{hd } (inouts_v''' \ x) = 0 \wedge \text{hd } (\text{tl } (inouts_v''' \ x)) = 0))$

assume $a2: ok_v''' \longrightarrow$
 $ok_v' \wedge$
 $(\forall xa. \text{length}(inouts_v''' \ x) = 2 \wedge$
 $\text{length}(inouts_v' \ x) = \text{Suc } 0 \wedge$
 $inouts_v' \ x = \text{take } (\text{Suc } 0) (inouts_v''' \ x) \bullet \text{drop } (\text{Suc } (\text{Suc } 0)) (inouts_v''' \ x) \wedge$
 $inouts_v''' \ x!(\text{Suc } 0) = x \ xa)$

assume $a3: \forall x. (\text{hd } (inouts_v \ x) = 0 \vee \text{hd } (inouts_v \ x) = 1) \wedge$
 $0 < inouts_v \ x!(\text{Suc } 0) \wedge$
 $inouts_v \ x!(\text{Suc } 0) < \text{max-door-open-time} \wedge$
 $(inouts_v \ x!\exists = 0 \vee inouts_v \ x!\exists = 1)$

assume $a4: \forall xa. \text{length}(inouts_v \ x) = 4 \wedge$
 $\text{length}(inouts_v'' \ x) = 5 \wedge$
 $inouts_v'' \ x = \text{take } 4 (inouts_v \ x) \bullet x \ xa \# \text{drop } 4 (inouts_v \ x)$

from $a4$ **have** $1: \forall xa. \text{length}(inouts_v \ x) = 4$
by *blast*

have $2: (\forall xa. (((\text{hd } (inouts_v \ x) \bullet [x \ xa]) = 0 \vee \text{hd } (inouts_v \ x) \bullet [x \ xa]) = 1) \wedge$
 $0 < (inouts_v \ x) \bullet [x \ xa]!(\text{Suc } 0) \wedge$
 $(inouts_v \ x) \bullet [x \ xa]!(\text{Suc } 0) < \text{max-door-open-time} \wedge$
 $((inouts_v \ x) \bullet [x \ xa]!\exists = 0 \vee (inouts_v \ x) \bullet [x \ xa]!\exists = 1)))$
 $= ((\text{hd } (inouts_v \ x) = 0 \vee \text{hd } (inouts_v \ x) = 1) \wedge$
 $0 < inouts_v \ x!(\text{Suc } 0) \wedge$
 $inouts_v \ x!(\text{Suc } 0) < \text{max-door-open-time} \wedge$
 $(inouts_v \ x!\exists = 0 \vee inouts_v \ x!\exists = 1))))$

using 1

by (*metis* *Suc-mono* *Suc-numeral* *hd-append2* *length-greater-0-conv* *nth-append* *numeral-2-eq-2*
numeral-3-eq-3 *semiring-norm(2)* *semiring-norm(8)* *zero-less-Suc*)

have $3: ok_v'''$
using 2 $a3$ $a1$ **by** *simp*

have $4: (\forall xa. \text{length}(inouts_v''' \ x) = 2 \wedge$
 $(\text{hd } (inouts_v \ x) = 1 \longrightarrow$
 $\text{hd } (inouts_v''' \ x) = 0 \wedge \text{hd } (\text{tl } (inouts_v''' \ x)) = 0))$
using 1 2 $a3$ $a1$ **by** (*smt* *hd-append2* *list.size(3)* *zero-neq-numeral*)

have $5: \forall xa. inouts_v' \ x = [\text{hd } (inouts_v''' \ x)]$

using 3 $a2$ **by** (*metis* *append-eq-conv-conj* *length-Cons* *list.size(3)* *list-equal-size2* *self-append-conv*)

show $ok_v' \wedge (\forall x. \text{length}(inouts_v' \ x) = \text{Suc } 0 \wedge (\text{hd } (inouts_v \ x) = 1 \longrightarrow \text{hd } (inouts_v' \ x) = 0))$
apply (*rule conjI*)
using 3 $a2$ **apply** *blast*

```

apply (rule allI)
apply (rule conjI)
using 3 a2 apply blast
using 3 a2 4 by (simp add: 5)
  qed
qed

```

Finally, the requirement is held for the *post-landing-finalize-1* because of transitivity of refinement relation.

```

lemma req-04:
  req-04-contract  $\sqsubseteq$  post-landing-finalize-1
  apply (simp only: post-landing-finalize-1-simp)
  using req-04-fd-ref req-04-ref by auto

```

end

References

- [1] MathWorks, “Simulink.” [Online]. Available: <https://uk.mathworks.com/products/simulink.html>
- [2] OSMC, “Openmodelica.” [Online]. Available: <https://openmodelica.org/>
- [3] R. D. Arthan, P. Caseley, C. O’Halloran, and A. Smith, “Clawz: Control laws in Z,” in *3rd IEEE International Conference on Formal Engineering Methods, ICFEM 2000, York, England, UK, September 4-7, 2000, Proceedings*. IEEE Computer Society, 2000, pp. 169–176.
- [4] P. Roy and N. Shankar, “Simcheck: a contract type system for simulink,” *Innovations in Systems and Software Engineering*, vol. 7, no. 2, p. 73, Jun. 2011. [Online]. Available: <http://dx.doi.org/10.1007/s11334-011-0145-4>
- [5] P. Boström and J. Wiik, “Contract-based verification of discrete-time multi-rate simulink models,” *Software and System Modeling*, vol. 15, no. 4, pp. 1141–1161, 2016.
- [6] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis, “Translating discrete-time simulink to lustre,” in *Embedded Software, Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003, Proceedings*, ser. Lecture Notes in Computer Science, R. Alur and I. Lee, Eds., vol. 2855. Springer, 2003, pp. 84–99.
- [7] A. Cavalcanti, P. Clayton, and C. O’Halloran, “Control law diagrams in Circus,” in *FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Newcastle, UK, July 18-22, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. S. Fitzgerald, I. J. Hayes, and A. Tarlecki, Eds., vol. 3582. Springer, 2005, pp. 253–268.
- [8] V. Preoteasa, I. Dragomir, and S. Tripakis, “The refinement calculus of reactive systems,” *CoRR*, vol. abs/1710.03979, 2017. [Online]. Available: <http://arxiv.org/abs/1710.03979>
- [9] F. Zeyda, J. Ouy, S. Foster, and A. Cavalcanti, “Formalising cosimulation models,” *Software Engineering and Formal Methods*, Jan. 2018. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-74781-1_31
- [10] B. Meyer, “Applying "design by contract",” *IEEE Computer*, vol. 25, no. 10, pp. 40–51, 1992.
- [11] C. B. Jones, *Wanted: a compositional approach to concurrency*. New York, NY: Springer New York, 2003, pp. 5–15. [Online]. Available: https://doi.org/10.1007/978-0-387-21798-7_1
- [12] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, “Moving from specifications to contracts in component-based design,” in *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, ser. Lecture Notes in Computer Science, J. de Lara and A. Zisman, Eds., vol. 7212. Springer, 2012, pp. 43–58.
- [13] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic, “Translating discrete-time simulink to lustre,” *ACM Trans. Embedded Comput. Syst.*, vol. 4, no. 4, pp. 779–818, 2005.

- [14] J. Woodcock and A. Cavalcanti, “A tutorial introduction to designs in unifying theories of programming,” in *Integrated Formal Methods*, E. A. Boiten, J. Derrick, and G. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 40–66.
- [15] C. Hoare and J. He, *Unifying theories of programming*. Prentice Hall, 1998, vol. 14.
- [16] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2283.
- [17] S. Foster, F. Zeyda, and J. Woodcock, “Isabelle/utp: A mechanised theory engineering framework,” in *Unifying Theories of Programming - 5th International Symposium, UTP 2014, Singapore, May 13, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, D. Naumann, Ed., vol. 8963. Springer, 2014, pp. 21–41.
- [18] M. Oliveira, A. Cavalcanti, and J. Woodcock, “A UTP Semantics for Circus,” *Formal Asp. Comput.*, vol. 21, no. 1-2, pp. 3–32, 2009.
- [19] J.-R. Abrial, *The B-book: assigning programs to meanings*. Cambridge University Press, 2005.
- [20] J. M. Spivey, *The Z Notation: A Reference Manual*, ser. Prentice Hall International Series in Computer Science. Prentice Hall, 1989.
- [21] N. Marian and Y. Ma, *Translation of Simulink Models to Component-based Software Models*. Forlag uden navn, 2007, pp. 274–280.
- [22] A. Cavalcanti, A. Mota, and J. Woodcock, “Simulink timed models for program verification,” in *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, ser. Lecture Notes in Computer Science, Z. Liu, J. Woodcock, and H. Zhu, Eds., vol. 8051. Springer, 2013, pp. 82–99.
- [23] A. Cavalcanti and J. Woodcock, “A tutorial introduction to CSP in *Unifying Theories of Programming*,” in *Refinement Techniques in Software Engineering, First Pernambuco Summer School on Software Engineering, PSSE 2004, Recife, Brazil, November 23-December 5, 2004, Revised Lectures*, ser. Lecture Notes in Computer Science, A. Cavalcanti, A. Sampaio, and J. Woodcock, Eds., vol. 3167. Springer, 2004, pp. 220–268.
- [24] C. A. R. Hoare and A. W. Roscoe, “Programs as Executable Predicates,” in *FGCS*, 1984, pp. 220–228.
- [25] V. Preoteasa and S. Tripakis, “Refinement calculus of reactive systems,” *CoRR*, vol. abs/1406.6035, 2014. [Online]. Available: <http://arxiv.org/abs/1406.6035>
- [26] S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, and F. Zeyda, “Unifying theories of reactive design contracts,” *In preparation for Theoretical Computer Science*, vol. abs/1712.10233, 2017.
- [27] I. Dragomir, V. Preoteasa, and S. Tripakis, “Compositional semantics and analysis of hierarchical block diagrams,” in *Model Checking Software - 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, The Netherlands, April 7-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, D. Bosnacki and A. Wijs, Eds., vol. 9641. Springer, 2016, pp. 38–56.

- [28] D. Bhatt, A. Chattopadhyay, W. Li, D. Oglesby, S. Owre, and N. Shankar, “Contract-based verification of complex time-dependent behaviors in avionic systems,” in *NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings*, ser. Lecture Notes in Computer Science, S. Rayadurgam and O. Tkachuk, Eds., vol. 9690. Springer, 2016, pp. 34–40.
- [29] VeTSS, “Uk research institute in verified trustworthy software systems.” [Online]. Available: <https://vetss.org.uk/>