Basic Two Controlled Tanks System

A case study to investigate the process to develop a formal and traceable paradigm to generate Requirements / Architectures / Models for System Design

# Contributors:

Aaron Fifarek – LinQuest Corporation supporting AFRL/RQQA

Kerianne Gross – AFRL/RQQA

Jon Hoffman – AFRL/RQQA

# Overview:

In the Verification and Validation of Complex Systems (VVCS) group within AFRL/RQQA early formalization of a system is critical to lessoning the effects of the growing cost and complexity of real world systems that the Air Force is currently and will need to certify today and in the future. It is well proven that many of the most costly errors found in modern systems originate from invalid requirements, incorrect design choices, and incorrect models of the system in question. The VVCS team maintains that if greater emphasis is taken to not only define a system in a formal semantic to limit knowledge loss between entities but to move analysis to these early stages then there will be capabilities to capture and mitigate these errors where it costs significantly less to resolve. [1] [2] [3]

In order to test the paradigm that requirements, architecture, and modeling are related and can be formalized and analyzed the team above, upon direction from Mr. Matt Clark (VVCS Technical Area Lead) was presented a multiple tank system based on the Laubwald paper [4] which describes a connected two tank system with a single inflow to a top tank linked to a second tank with two output valves. This is the first iteration of this team’s representation of this system and it contains and number of assumptions to simplify the problem such as instantaneous switching of actuator states and simplified liquid flow dynamics. The complete representation of this system is included in the formally defined requirements document (Appendix A). The content of the package is separated into three related sections: 1) Requirements, 2) Architecture, and 3) Model.

In the project is included two related representations of the defined system. The first is the *functional* representation of the system. This system representation is based off of a functional representation of the system in AGREE and in Simulink. Specifically, Boolean values and standard Simulink block were used to generate the representations of these aspects of the design. The other representation is the *primitive* representation of the system. This system utilizes AGREE to generate verifiable *primitives* of Simulink blocks. These blocks are proved to function as defined through AGREE analysis of the block behaviors. These representations are closely related with some changes to compensate for the proved AGREE primitives as well as challenges posed by the use of SLDV. It is important to note that the SLDV package is SDLV\_2012.

# Content Summary:

* Requirements
  + SpeAR Functional
    - Coupled Tanks Requirements in SpeAR (Appendix B)
  + Coupled Tanks Requirements Document [Basic] (Appendix A)
* Architecture
  + AGREE Functional Design
    - Coupled Tanks Architecture in OSATE/AGREE (Appendix C)
  + AGREE Primitive Design
    - Coupled Tanks Architecture in OSATE/AGREE (Appendix D)
* Model
  + Simulink Functional
    - Funct\_Sim\_12B\_qvt\_00.mdl (Appendix E, Figure 1)
  + Simulink Primitive
    - Prim\_Sim\_12B\_qvt.slx (Appendix E, Figure 2)
  + SDLV Primitive
    - Coupled Tanks Simulink Files Description (Appendix F)
    - Project files to execute with SLDV

# Acronyms:

AGREE Assume Guarantee REasoning Environment

OSATE Open Source AADL Tool Environment

SLDV Simulink Design Verifier

SpeAR Specification and Analysis for Requirements

# References

|  |  |
| --- | --- |
| [1] | B. W. Boehm, Software Engineering Economics, Prentice Hall, 1981. |
| [2] | D. Galin, Software Quality Assurance: From Theory to Implementation, Pearson/Addison-Wesley, 2004. |
| [3] | NIST, "The Economic Impacts of Inadequate Infrastructure for Software Testing," 2002. |
| [4] | E. Laubwald, "Coupled tanks systems 1," in *Control Systems Principles*, 2006. |

# Appendix A

This is the requirements document for the Basic Control Tanks System

Contents

[Introduction 2](#_Toc410900937)

[System Description 3](#_Toc410900938)

[System Level Assumptions 6](#_Toc410900939)

[1. Inflow 6](#_Toc410900940)

[2. Outflow 6](#_Toc410900941)

[System Level Requirements 6](#_Toc410900942)

[High Level Requirements 6](#_Toc410900943)

[Primary Subsystems 7](#_Toc410900944)

[1. Environment 7](#_Toc410900945)

[1.1 System Time Increment Assumptions 7](#_Toc410900946)

[1.2 Tank 1 Size Assumptions 7](#_Toc410900947)

[1.3 Tank 2 Size Assumptions 7](#_Toc410900948)

[1.4 Previous Height Assumptions 8](#_Toc410900949)

[1.5 Control Input Assumptions 8](#_Toc410900950)

[1.6 Sensor Requirements 9](#_Toc410900951)

[1.7 Tank 1 Requirements 10](#_Toc410900952)

[1.8 Tank 2 Requirements 12](#_Toc410900953)

[2. Controller 13](#_Toc410900954)

[2.1 Tank 1 Controller 13](#_Toc410900955)

[2.2 Tank 2 Controller 14](#_Toc410900956)

[Simulation Requirements 15](#_Toc410900957)

[1. Updater 15](#_Toc410900958)

[Mode State Transitions 17](#_Toc410900959)

[Mode Diagrams 17](#_Toc410900960)

[1.1 Tank 1 Modes: 17](#_Toc410900961)

[1.2 Tank 2 Modes: 17](#_Toc410900962)

[Appendix A – Environmental Constants 18](#_Toc410900963)

# Introduction

The Verification and Validation of Autonomous Systems (VVAS) Team at the Air Force Research Laboratory’s Aerospace Systems Directorate seeks to expand the current design paradigm to incorporate more verification and validation techniques early in system design in order to generate evidence throughout the design process and supplement traditional modeling, simulation, test, and evaluation techniques. This approach to certification is called an Assurance Case[[1]](#footnote-1) and it is envisioned to be a compilation of analytical proofs, comprehensive test, and bounding of an advanced system at runtime presented to make the case for system certification.

The vision for incorporating verification and validation techniques early in design is to start by formally expressing requirements, architecture, and modeling development in a compositional structure and iterative process that employs formal methods to generate analytical proofs and reason about the design at the earliest stages when discovering errors is the least costly. Rather than creating monolithic designs, systems will be logically broken out by components with analysis completed at each level of the design.

This document is an example requirements document for this new paradigm. As the design process is envisioned to be iterative, this requirements document reflects a later, more mature iteration in which some elements of the design will be reflected in derived requirements in order to produce a more complete picture and foundation for analysis.

Requirements documented here are also written formally in a tool developed by Rockwell Collins and AFRL called Specification and Analysis of Requirements (SpeAR). The SpeAR framework allows requirements to be analyzed using model checking and theorem proving. These requirements are used to analyze the defined architecture and model of the system in the next two design phases.

In order to demonstrate the assurance case concept and evaluate available tools for the early system design, a publically releasable, academic control example simple enough to completely design and evaluate in software, but with enough richness to expand with complex equations, mode logic, advanced control algorithms, fault tolerance, etc. was desired. After evaluating a couple options, the coupled tank system was selected as a popular academic example used in control theory exercises with potential application to the F-35 program. This is the first iteration of this system design and seeks only to define very basic functionality with many simplifying assumptions included.

The document provides a description of the coupled tanks system used in this proof of concept assurance case as well as a list of requirements.

# System Description

The coupled tank system utilized draws liquid of some form from a limitless source, temporarily stores it in order for some process such as mixing or temperature control to occur, and then releases the liquid into a bottomless sink. As the name would suggest, the system contains two tanks that are each controlled by one of two independent controllers that operate four actuators between the two tanks and receive feedback from five sensor dispersed between the two tanks that provide a rudimentary understanding of the state of the liquid height in each tank. The system is implemented in the architecture phase with the physical definition presented in Figure 1. The interaction between the physical design and that of the system controller is presented in Figure 2.

|  |  |
| --- | --- |
| Figure 1: Coupled Tanks Physical Architecture | Figure 2: Coupled tanks physical sensor and actuator signal connections to the system controller. |

Liquid is drawn into Tank 1 by a simple on/off pump and expelled from Tank 1 by a simple on/off valve, which empties liquid directly into Tank 2. Tank 1 which is 7.0 meters high and 1.0 square meter in cross-sectional area. The objective of the Tank 1 controller is to keep the height of the liquid between the High Sensor (SH) and the Low Sensor (SL) in the “safe” region by controlling the state of the pump and valve.

Liquid enters Tank 2 by the valve from Tank 1 and exits via a production valve or a larger emergency valve when a larger flow out is required for the system to maintain safety. Tank 2 is 4.0 meters in height and 2.0 square meters in cross sectional area. The objective of the Tank 2 controller is to keep the height of the liquid in tank two in the “safe” region monitored by the position of a high (SH), medium (SM), and low (SL) sensor in Tank2 by controlling the state of the production and emergency valves. Control of the input valve rests only with the Tank 1 controller, so Tank 2 is a reactionary controller to the input from Tank 1.

It is assumed that the flow rate through the pump and three valves is either zero or a constant value, that no faults or leaks occur in the system, and there are not external disturbance inputs to the coupled tank system. All sensors used in this system are assumed to be Boolean sensors that report 1 if the liquid is at the sensor height and 0 if the liquid is not at the sensor’s height. Simple components such as the actuators and sensors in this example might be selected to reduce cost and opportunity for the system to malfunction.

The system is organization in a functional composition as depicted in Figure 3:

Figure 3: Functional Compositional Architecture of the Coupled Tanks System

The Coupled Tanks System is broken down from the highest level into a controller and environment. The controller is then broken down into a separate controller for each tank. The environment is broken down functionally into each type of component (Tanks, Valves, Pumps, and Sensors) and then broken down even further from there into the individual components. This structure allows analysis to be completed at the lowest levels of the design with results of the proof flowing up to higher levels of the design until the entire system is analyzed. This is in contrast to monolithic approaches that only analyze the system as a whole, which presents a large computational burden and makes it difficult to identify particular faults in the system design.

# System Level Assumptions

## Inflow

###### There is an unlimited supply of water to be pumped into the coupled tanks system

###### The water is not pressurized in the production line and cannot be gravity fed into the coupled tanks system, it must be pumped into the coupled tanks system

###### The system shall contain a pump to draw liquid into tank 1.

## Outflow

###### There is an unlimited receptacle to receive an unlimited quantity of liquid emptied from the production valve.

###### There is an unlimited receptacle can receive an unlimited amount of liquid emptied from the emergency valve.

###### The system shall contain a valve to transport liquid from tank 1 to tank 2.

###### The system shall contain a valve to transport liquid from tank 2 to the production line

###### The system shall contain a valve to transport liquid from tank 2 to an acceptable area outside the production system in case of emergency.

# System Level Requirements

## High Level Requirements

#### Tank 1 shall not overflow.

Rationale: An overflow of the tank is a safety hazard.

#### Tank 2 shall not overflow.

Rationale: An overflow of the tank is a safety hazard.

#### The initial liquid height Tank 1 shall be known to the control system.

Rationale: In order to properly track the liquid height in Tank 1, the initial liquid height must be known. This system will meet this requirement by starting from a completely empty state or some initial known height below the low sensor height.

#### The initial liquid height Tank 2 shall be known to the control system.

Rationale: In order to properly track the liquid height in Tank 2, the initial liquid height must be known. This system will meet this requirement by starting from a completely empty state or some initial known height below the low sensor height.

#### The height of the tank 1 liquid shall not be over the tank 1 high safety level for more than the specified time. This specified time is dependent on the flowrate per timestep.

Rationale: Prolonged periods of time where the liquid is higher than the safety level could cause an unsafe situation to occur. This is dependent on the specified time increment per timestep. A violation will be identified but will not change the pump and valve states until a following timestep. Therefore the number of allowable timesteps that the system can violate the safety boundary before returning to the safe region is dependent on the system architecture.

#### The height of the tank 1 liquid shall not be lower than the tank 1 low safety level (once it has already crossed it) for more than the specified time. This specified time is dependent on the flowrate per timestep.

Rationale: Prolonged periods of time where the liquid is lower than the safety level could cause an unsafe situation to occur. This excludes the initial startup period while the tank is filling for the first time. This is dependent on the specified time increment per timestep. A violation will be identified but will not change the pump and valve states until a following timestep. Therefore the number of allowable timesteps that the system can violate the safety boundary before returning to the safe region is dependent on the system architecture.

#### The height of the tank 2 liquid shall not be over the tank 2 high safety level for more than the specified time. This specified time is dependent on the flowrate per timestep.

Rationale: Prolonged periods of time where the liquid is higher than the safety level could cause an unsafe situation to occur. This is dependent on the specified time increment per timestep. A violation will be identified but will not change the pump and valve states until a following timestep. Therefore the number of allowable timesteps that the system can violate the safety boundary before returning to the safe region is dependent on the system architecture.

#### The height of the tank 2 liquid shall not be lower than the tank 2 low safety level (once it has already crossed it) for more than the specified time. This specified time is dependent on the flowrate per timestep.

Rationale: Prolonged periods of time where the liquid is lower than the safety level could cause an unsafe situation to occur. This excludes the initial startup period while the tank is filling for the first time. This is dependent on the specified time increment per timestep. A violation will be identified but will not change the pump and valve states until a following timestep. Therefore the number of allowable timesteps that the system can violate the safety boundary before returning to the safe region is dependent on the system architecture.

# Primary Subsystems

## Environment

The purpose of the environment subsystem is to represent the physical attributes of the system. Specifically this subsystem includes all the tanks, actuators, pumps, valves, and sensors that is necessary to allow the system to exist in a physical representation. This subsystem produces changes in component states only in response to the physical attributes of the tanks.

### System Time Increment Assumptions

###### The default time increment of the system, once defined, does not change.

Rationale: The time increment represents the global polling rate of the physical system and is not variable. The rate is defined globally and governs represents the rate that the system can check the dynamic state of the system.

### Tank 1 Size Assumptions

###### The tank 1 has a specified volume that is known globally and does not change.

Rationale: The tank 1 specified volume is known so that system can calculate the expected volume in the tank depending on inflow and outflow calculations.

###### The tank 1 has a specified cross-sectional area that is known to the controller and does not change.

Rationale: The tank 1 specified cross-sectional area is known so that system can calculate the expected volume in the tank depending on inflow and outflow calculations. Furthermore, this information provides the ability determine liquid height in tank 1by volume or to determine the tank volume by the liquid height.

###### The tank 1 has a specified height that is known to the controller and does not change.

Rationale: The tank 1 specified height is known so that the system can determine whether or not an overflow of tank 1 has occurred.

### Tank 2 Size Assumptions

###### The tank 2 has a specified volume that is known to the controller and does not change.

Rationale: The tank 2 specified volume is known so that system can calculate the expected volume in the tank depending on inflow and outflow calculations.

###### The tank 2 has a specified cross-sectional area that is known to the controller and does not change.

Rationale: The tank 2 specified cross-sectional area is known so that system can calculate the expected volume in the tank depending on inflow and outflow calculations. Furthermore, this information provides the ability determine liquid height in tank 2 by volume or to determine the tank volume by the liquid height.

###### The tank 2 has a specified height that is known to the controller and does not change.

Rationale: The tank 2 specified height is known so that system can determine whether or not an overflow of tank 2 has occurred.

### Previous Height Assumptions

###### The previous liquid height of tank 1 is the liquid height at the previous time step.

Rationale: The previous liquid height in tank 1 is defined as the liquid height at the previously polled timestep. The initial liquid height of the tank is below the low sensor and is known.

###### The previous liquid height of tank 2 is the liquid height at the previous time step.

Rationale: The previous liquid height in tank 2 is defined as the liquid height at the previously polled timestep. The initial liquid height of the tank is below the low sensor and is known.

### Control Input Assumptions

###### The pump is initially off when the system is started.

Rationale: The initial state of the pump is necessary to ensure that the system does not have any ability to fill or drain that is not under control.

###### The tank 1 outflow valve is initially closed.

Rationale: The tank 1 should not drain unless controlled to do so. This sets the understood initial state of the tank 1 outflow valve.

###### The tank 2 production valve is initially closed.

Rationale: The tank 2 should not drain liquid into the production line unless controlled to do so. This sets the understood initial state of the tank 2 production valve.

###### The tank 2 emergency valve is initially closed.

Rationale: The tank 2 should not drain liquid into through the emergency valve unless controlled to do so. This sets the understood initial state of the tank 2 emergency valve.

###### The system liquid sensors shall be placed properly in each tank to match the predefined sensor height locations.

Rationale: All tank liquid sensors will be placed in the tank at predefined heights and those heights will be globally known to the overall system. The physical placement of these sensors will match the system configured placement.

###### When the pump is ON, the pump shall maintain a maximum flowrate that is known to the system.

Rationale: The pump into the tank 1 will provide a maximum inflow to that is defined in the system. Other subsystems will use this value when requesting the maximum inflow into tank 1.

###### The valves in the system shall have a constant maximum flowrate that is known to the system.

Rationale: All the valves in the system will have globally known maximum flowrates that can be used by other subsystems in order to estimate the current liquid heights in the tanks.

### Sensor Requirements

#### The tank 1 HIGH sensor is physically placed at a greater height (above) the tank 1 LOW sensor.

Rationale: The HIGH sensor is assumed to be placed at a height level within tank 1 that would signal a greater height of liquid in tank 1 than that of the LOW sensor.

#### The tank 2 HIGH sensor is physically placed at a greater height (above) the tank 2 MID sensor.

Rationale: The HIGH sensor is assumed to be placed at a height level within tank 2 that would signal a greater height of liquid in the tank than that of the MID sensor.

#### The tank 2 MID sensor is physically placed at a greater height (above) the tank 2 LOW sensor.

Rationale: The MID sensor is assumed to be placed at a height level within tank 2 that would signal a greater height of liquid in tank 2 than that of the LOW sensor.

#### The sensor outputs shall initially be false.

Rationale: Tank 1 and tank 2 will initially have no liquid (empty) or be filled to a level below low sensor heights and therefore the liquid height sensors in the system will provide a false, or 0, initial output.

#### If the tank 1 liquid height is greater than or equal to the sensor height of the tank 1 HIGH liquid sensor then the sensor should return an active (TRUE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid reaches that defined height of the sensor then it returns a signal that informs the system.

#### If the tank 1 liquid height is below (less than) the placement height of the tank 1 HIGH liquid sensor then the sensor should return a (FALSE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid does not reach that defined height of the sensor then a lack of a signal informs the system water is not present at that level.

#### If the tank 1 liquid height is greater than or equal to the placement height of the tank 1 LOW liquid sensor then the sensor should return an active (TRUE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid reaches that defined height of the sensor then it returns a signal that informs the system.

#### If the tank 1 liquid height is below (less than) the placement height of the tank 1 LOW liquid sensor then the sensor should return a (FALSE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid does not reach that defined height of the sensor then a lack of a signal informs the system water is not present at that level.

#### If the tank 2 liquid height is greater than or equal to the placement height of the tank 2 HIGH liquid sensor then the sensor should return an active (TRUE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid reaches that defined height of the sensor then it returns a signal that informs the system.

#### If the tank 2 liquid height is below (less than) the placement height of the tank 2 HIGH liquid sensor then the sensor should return a (FALSE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid does not reach that defined height of the sensor then a lack of a signal informs the system water is not present at that level.

#### If the tank 2 liquid height is greater than or equal to the placement height of the tank 2 MID liquid sensor then the sensor should return an active (TRUE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid reaches that defined height of the sensor then it returns a signal that informs the system.

#### If the tank 2 liquid height is below (less than) the placement height of the tank 2 MID liquid sensor then the sensor should return a (FALSE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid does not reach that defined height of the sensor then a lack of a signal informs the system water is not present at that level.

#### If the tank 2 liquid height is greater than or equal to the placement height of the tank 2 LOW liquid sensor then the sensor should return an active (TRUE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid reaches that defined height of the sensor it will return a signal that informs the system.

#### If the tank 2 liquid height is below (less than) the placement height of the tank 2 LOW liquid sensor then the sensor should return a (FALSE) state to the system.

Rationale: The sensors are binary switches that trigger when the liquid has reached the sensor height. If the liquid does not reach that defined height of the sensor then a lack of a signal informs the system water is not present at that level.

### Tank 1 Requirements

#### The tank 1 is initially empty or below the low tank 1 liquid height sensor. If the tank is not empty the liquid height must be known to the system.

Rationale: The system is designed to have a specified start condition to sync the controller and that environment.

#### The tank 1 change in liquid volume for each timestep is limited by the maximum inflow and outflow for each timestep.

Rationale: Tank 1 can only fill or empty by the properties of the overall inflow and outflow from the tank which is limited by the state of the pump and valve. The tank liquid height must follow the physical properties of these components.

#### The tank 1 resulting liquid height for each timestep is dependent on the liquid height in the previous timestep plus the additional liquid height defined by the net flow rate from the state of the pump and the valve and the cross-sectional area of tank 1.

Rationale: Tank 1 can only fill or empty dependent on the pump and valve state. Each timestep would allow a delta height into the tank which will change the liquid height in the previous timestep.

#### When the tank 1 pump is ON and the valve is CLOSED the liquid height in the tank shall increase from the previous timestep.

Rationale: When the tank 1 pump is providing an inflow of liquid into the tank and the valve is closed, which prevents any outflow of liquid from this tank, the liquid height in the tank will increase (rise).

#### When the tank 1 pump is OFF and the valve is OPEN the liquid height in the tank shall decrease from the previous timestep.

Rationale: When the tank 1 pump has no inflow of liquid into the tank and the valve is open, which drains liquid from this tank, the liquid height in the tank will decrease (drop).

#### When the tank 1 pump is OFF that the valve is CLOSED the liquid height in the tank shall remain equivalent to the liquid height from the previous timestep.

Rationale: When the tank 1 pump has no inflow of liquid into the tank and the valve is closed, which prevents any outflow of liquid from this tank, the liquid height in the tank will be equal to the height at the previous timestep (remain steady).

#### When the pump is ON there should be inflow into the tank 1.

Rationale: When this pump is on then an inflow of liquid should be added to tank 1. The tank can still have a negative flowrate depending on the states of the outflow properties of the tank.

#### When the pump is OFF there should be NO inflow into tank 1.

Rationale: When this pump is off then can be no inflow of liquid added to tank 1.

#### When the tank 1 valve is OPEN the flow rate shall be equal to the physical maximum flow rate of the valve.

Rationale: When this valve is open then liquid can flow from tank 1 into tank 2. This will only hold as long as there is available liquid in tank 1 to transfer to tank 2. In the event that tank 1 is empty then there would be no flow to the tank 2.

#### When the tank 1 valve is CLOSED the flow rate of the valve shall be zero.

Rationale: When this valve is closed then liquid cannot flow from tank 1 into the tank 2.

### Tank 2 Requirements

#### Tank 2 is initially empty or below the tank 2 low liquid height sensor. If the tank is not empty the liquid height must be known to the system.

Rationale: The system is designed to have a specified start condition to sync the controller and that environment.

#### The tank 2 change in liquid volume for each timestep is limited by the maximum inflow and outflow for each timestep.

Rationale: Tank 2 can only fill or empty by the properties of the overall inflow and outflow from the tank which is limited by the state of the inflow valve and the production and emergency outflow valves. The tank liquid height must follow the physical properties of these components.

#### The resulting tank 2 liquid height for each timestep is a function of the liquid height in the previous timestep, the additional liquid height as defined by the net flow rate from the state of the inflow valve from tank 1 and tank 2 outflow valves (production and emergency), and the cross-sectional area.

Rationale: Tank 2 can only fill or empty dependent on the inflow valve state and the outflow by the production and emergency valve states. Each timestep would allow a delta height into the tank which will change the liquid height in the previous timestep.

#### When the tank 1 valve is OPEN and the tank 2 outflow valves (emergency and production) are CLOSED the liquid height in tank 2 shall increase from the previous timestep.

Rationale: When the tank 1 valve is providing an inflow of liquid into tank 2 and the tank 2 valves (production and emergency) are closed, which prevents any outflow of liquid from tank 2, the liquid height in the tank 2 will increase (rise).

#### When the tank 1 valve is OPEN, the tank 2 production valve is OPEN, and the tank 2 emergency valve is CLOSED the liquid height in tank 2 shall increase from the previous timestep.

Rationale: When the tank 1 valve is providing an inflow of liquid into tank 2, the tank 2 production valve is open (outflow), and the tank 2 emergency valve is closed, the tank 2 liquid height will increase (rise). This will not hold if the production valve flowrate is greater than or equal to the flowrate of the valve from the tank 1.

#### When the tank 1 valve is OPEN, the tank 2 production valve is CLOSED, and the tank 2 emergency valve is OPEN the liquid height in tank 2 shall decrease from the previous timestep.

Rationale: When the tank 1 valve is providing an inflow of liquid into tank 2, while the tank 2 production valve is closed, and while the tank 2 emergency valve is open (outflow), the tank 2 liquid height will decrease (fall). This will not hold if the emergency valve flowrate is less than or equal to the flowrate of the valve from tank 1.

#### When the tank 1 valve is CLOSED that the tank 2 valves (production and emergency) are CLOSED the liquid height in tank 2 shall remain the same from the previous timestep.

Rationale: When the tank 1 valve provides no inflow of liquid into tank 2and the tank 2 valves are closed, which prevents any outflow of liquid from this tank, the liquid height in tank 2 will be equal to the height at the previous timestep (remain steady).

#### When the tank 2 production valve is OPEN there should be an outflow equal to that of the max production flow.

Rationale: While the production valve in tank 2 is open the flow through that valve should be equal to the max flowrate through the valve as defined by the physical attributes of that valve.

#### When the tank 2 production valve is CLOSED there should be no flow from that valve.

Rationale: While the production valve in the tank 2 is closed there should be no flow through the valve since it is in engaged in a blocked state.

#### When the tank 2 emergency valve is OPEN there should be an outflow equal to that of the max emergency output flow.

Rationale: While the emergency valve in tank 2 is open the flow through that valve should be equal to the max flowrate through the valve as defined by the physical attributes of that valve.

#### When the tank 2 emergency valve is CLOSED there should be no flow from that valve.

Rationale: While the emergency valve in tank 2 is closed there should be no flow through the valve since it is in engaged in a blocked state.

## Controller

The purpose of the controller subsystem is to provide the control responses to the system to manage the liquid level of the tanks through the use of the pump, valves, and sensors. The controller is provided information from the physical system in the manner that the physical system can provide. This subcomponent provides the logic in response to the signals provided by the physical subsystem.

### Tank 1 Controller

###### When the tank 1 HIGH sensor provides a TRUE signal to the controller, it is understood that the tank 1 LOW sensor is also TRUE.

Rationale: If the liquid level has reached the high sensor in tank 1 and physically (as defined in the environment) the low sensor is at lesser height (lower) in the tank than if the water reaches the high sensor it must also exceed the height of the low sensor height and therefore should also be TRUE.

#### When the tank 1 LOW liquid sensor is FALSE, the tank 1 pump shall be ON and the tank 1 outflow valve shall be CLOSED.

Rationale: This requirement defines an operational behavior of tank 1. In the event that the safety low sensor level is not reached (water level too low) then the controller will not allow any liquid to drain from the tank until the water volume has filled to an appropriate level.

#### When the tank 1 HIGH liquid sensor is TRUE, the tank 1 pump shall be OFF and the tank 1 outflow valve shall be OPEN.

Rationale: This requirement defines an operational behavior of tank 1. In the event that the safety high sensor level is reached or exceeded (water level too high) then the controller will drain liquid from the tank until the water volume has filled to an appropriate level. This is accomplished by turning off the pump and opening the outflow valve, thus removing inflow and maximizing outflow.

#### When the tank 1 HIGH liquid sensor is FALSE and the tank 1 LOW liquid sensor is TRUE then the tank 1 pump and the tank 1 outflow valve shall maintain the states they were set to in the previous timestep.

Rationale: This requirement defines an operational behavior of tank 1. In the event that the safety high sensor level is reached or exceeded (water level too high) then the controller will drain liquid from the tank until the water volume has filled to an appropriate level. This is accomplished by turning off the pump and opening the outflow valve, thus removing inflow and maximizing outflow.

### Tank 2 Controller

###### When the tank 2 HIGH sensor provides a TRUE signal to the controller, it is understood that the tank 2 MID sensor is also TRUE.

Rationale: If the liquid level has reached the high sensor in tank 2, the mid sensor is at lesser height (lower) in the tank. Therefore if the water reaches the high sensor it must also exceed the height of the mid sensor height.

###### When the tank 2 MID sensor provides a TRUE signal to the controller, it is understood that the tank 2 LOW sensor is also TRUE.

Rationale: If the liquid level has reached the mid sensor in tank 2, the low sensor is at lesser height (lower) in the tank. Therefore if the water reaches the mid sensor it must also exceed the height of the low sensor height.

#### When the system is initially started the tank 2 production valve shall be CLOSED.

Rationale: In order to prevent the production line from receiving liquid from the system prior to the completion of startup, the production valve is closed at the initial startup of the system.

#### When the system is initially started the tank 2 production valve shall be CLOSED.

Rationale: In order to prevent the production line from receiving liquid from the system prior to the completion of startup, the production valve is closed at the initial startup of the system.

#### When the system is initially started the tank 2 emergency valve shall be CLOSED.

Rationale: The emergency valve is closed at the initial startup of the system in order to ensure that the system does not start in a state where liquid can leave the tanks without a command by the tank controllers.

#### When the tank 2 LOW sensor is FALSE, the emergency valve and the production valve (outflow valves) shall be commanded to be CLOSED.

Rationale: When the tank 2 low sensor is FALSE that signal informs the tank 2 controller that the liquid in tank 2 has not yet met the minimum height requirement to allow outflow from the tank. Therefore all outflow valves are closed.

#### When the tank 2 LOW sensor is TRUE and the tank 2 MID sensor is FALSE, then emergency valve shall be commanded to be CLOSED and the production valve shall be commanded to be OPEN.

Rationale: When the tank 2 low sensor is TRUE and the tank 2 mid sensor is FALSE, those signals inform the tank 2 controller that the liquid in tank 2 has not yet met the mid tank height sensor but has exceeded the low sensor tank height. This provides the controller knowledge at which height band in the tank is the liquid level.

#### When the tank 2 HIGH sensor is TRUE, the emergency valve and the production valve (outflow valves) shall be commanded to be OPEN.

Rationale: When the tank 2 high sensor is TRUE, that signal informs the tank 2 controller that the liquid in tank 2 has met or exceeded the maximum height requirement in tank 2. Therefore all outflow valves are open.

#### When the tank 2 MID sensor is TRUE, the tank 2 HIGH sensor is FALSE, and the emergency valve was previously open, then the emergency valve and the production valve (outflow valves) shall be commanded to be OPEN.

Rationale: When the tank 2 mid sensor is TRUE, the tank 2 high sensor is false, and that emergency valve was previously open means that tank 2 had been in a exceed high sensor state at a previous timestep. In this situation the tank should empty until the mid sensor is no longer active (TRUE) to eliminate the emergency fill height situation.

#### When the tank 2 MID sensor is TRUE, the tank 2 HIGH sensor is FALSE, and the emergency valve was not previously open, then the emergency valve shall remain commanded CLOSED and the production valve shall be commanded to be OPEN.

Rationale: When the tank 2 mid sensor is TRUE, the tank 2 high sensor is false, and that emergency valve was not previously open means that tank 2 liquid level is in the High-Mid liquid band. This does not represent an emergency drain state and therefore should just continue to outflow to the production line as expected.

# Simulation Requirements

In order to design a simulation of the described system it is necessary to add an additional primary subsystem to the design. This subsystem is responsible for updating the state variables that track the current and previous states of the system.

## Updater

This subsystem provides the system an ability to have a history of the previous state of the system as defined by the global time increment. This subsystem provides this data so that the controller subsystem can use the data to calculate expected physical data of the environment.

#### The Tank 1 Previous Height shall initially be the specified initial value.

Rationale: The Tank 1 Previous Height should be initialized correctly.

#### The Tank 2 Previous Height shall initially be the specified initial value.

Rationale: The Tank 2 Previous Height should be initialized correctly.

#### The Previous Pump State shall initially be the specified initial value.

Rationale: The Previous Pump State should be initialized correctly.

#### The Previous Valve State shall initially be the specified initial value.

Rationale: The Previous Valve State should be initialized correctly.

#### The Previous Production Valve State shall initially be the specified initial value.

Rationale: The Previous Production Valve State shall be initialized correctly.

#### The Previous Emergency Valve State shall initially be the specified initial value.

Rationale: The Previous Emergency Valve State shall be initialized correctly.

#### After the initial time step, the Tank 1 Previous Height shall be equal to the Tank 1 Height in the previous step.

Rationale: The Tank 1 Previous Height should be updated to take the value of the Tank 1 Height in the previous step.

#### After the initial time step, the Tank 2 Previous Height shall be equal to the Tank 2 Height in the previous step.

Rationale: The Tank 2 Previous Height should be updated to take the value of the Tank 2 Height in the previous step.

#### After the initial time step, the Previous Pump State shall be equal to the Pump State in the previous step.

Rationale: The Previous Pump State should be updated to take the value of the Pump State in the previous step.

#### After the initial time step, the Previous Valve State shall be equal to the Valve State in the previous step

Rationale: The Previous Valve State should be updated to take the value of the Valve State in the previous step.

#### After the initial time step, the Previous Production Valve State shall be equal to the Production Valve State in the previous step.

Rationale: The Previous Production Valve State shall be updated to take the value of the Production Valve State in the previous step.

#### After the initial time step, the Previous Emergency Valve State shall be equal to the Emergency Valve State in the previous step.

Rationale: The Previous Emergency Valve State shall be updated to take the value of the Emergency Valve State in the previous step.

# Mode State Transitions

This section defines the designed mode logic of the tank controllers in the system.

## Mode Diagrams

### Tank 1 Modes:

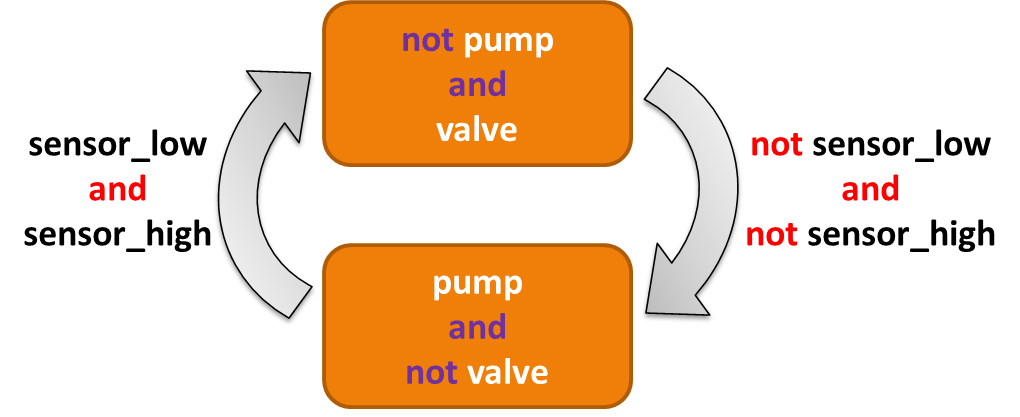


Figure 4: Tank 1 mode logic state machine

### Tank 2 Modes:

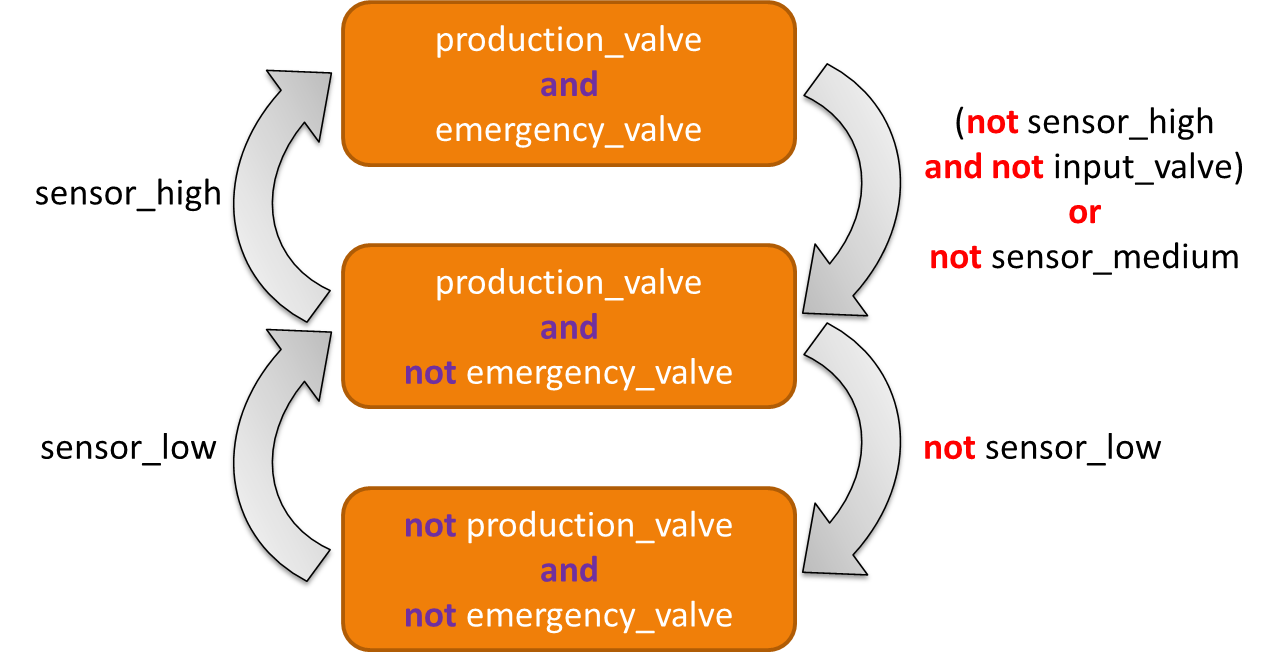


Figure 5: Tank 2 mode logic state machine

# Appendix A.1 – Data Tables

Table 1: Environmental constants used in the System Simulation

|  |  |  |
| --- | --- | --- |
| Environment Constants | Value | Units |
| Time\_constant | 0.1 | s |
| Tank1\_Sensor\_High\_Height | 5.0 | m |
| Tank1\_Sensor\_Low\_Height | 2.0 | m |
| Tank1\_Pump\_Flow\_Rate | 0.2 | m3/s |
| Tank1\_Valve\_Flow\_Rate | 0.1 | m3/s |
| Tank1\_Cross\_Section\_Area | 1.0 | m2 |
| Tank1\_Max\_Volume | 7.0 | m3 |
| Tank2\_Sensor\_High\_Height | 3.0 | m |
| Tank2\_Sensor\_Mid\_Height | 2.0 | m |
| Tank2\_Sensor\_Low\_Height | 1.0 | m |
| Tank2\_Production\_Valve\_Flow\_Rate | 0.05 | m3/s |
| Tank2\_Emergency\_Valve\_Flow\_Rate | 0.2 | m3/s |
| Tank2\_Cross\_Section\_Area | 2.0 | m2 |
| Tank2\_Max\_Volume | 8.0 | m3 |

Table 2: Summary of the initial state of the system to be maintained in the physical and simulated design

|  |  |  |
| --- | --- | --- |
| Initial State of System Components | Value | Units |
| Pump State | Off | - |
| Tank1 Valve State | Closed | - |
| Tank1 Liquid Height (if empty) | 0.0 | m |
| Tank1 Liquid Height (if not empty) | 0.0 < x < Tank1\_Sensor\_Low\_Height | m |
| Tank1\_Sensor\_High | Inactive (Off) | - |
| Tank1\_Sensor\_Low | Inactive (Off) | - |
| Tank2 Liquid Height (if empty) | 0.0 | m |
| Tank2 Liquid Height (if not empty) | 0.0 < x < Tank2\_Sensor\_Low\_Height | m |
| Tank2\_Sensor\_High | Inactive (Off) | - |
| Tank2\_Sensor\_Mid | Inactive (Off) | - |
| Tank2\_Sensor\_Low | Inactive (Off) | - |
| Tank2 Production Valve | Closed | - |
| Tank2 Emergency Valve | Closed | - |

# Appendix B

This is the source code for the SpeAR functional representation of the Basic Control Tanks System

==============================================================================

.project

==============================================================================

<?xml version="1.0" encoding="UTF-8"?>

<projectDescription>

<name>Ctrl\_Tanks\_Basic</name>

<comment></comment>

<projects>

</projects>

<buildSpec>

<buildCommand>

<name>org.eclipse.xtext.ui.shared.xtextBuilder</name>

<arguments>

</arguments>

</buildCommand>

</buildSpec>

<natures>

<nature>org.eclipse.xtext.ui.shared.xtextNature</nature>

</natures>

</projectDescription>

==============================================================================

Controller.spear

==============================================================================

Relation Ctrl\_Tanks.Controller;

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.tank1\_ctrl\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.tank2\_ctrl\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.controller\_output\_type.\*;

Inputs:

t1\_sensor\_high : bool;

t1\_sensor\_low : bool;

t2\_sensor\_high : bool;

t2\_sensor\_mid : bool;

t2\_sensor\_low : bool;

Outputs:

ctrl\_output : controller\_output\_type;

State:

t1\_ctrl : tank1\_ctrl\_output\_type;

t2\_ctrl : tank2\_ctrl\_output\_type;

Macros:

tank1\_ctrl\_pump\_state : bool = t1\_ctrl{pump\_state\_cmd};

tank1\_ctrl\_valve\_state : bool = t1\_ctrl{valve\_state\_cmd};

tank2\_ctrl\_p\_valve\_state : bool = t2\_ctrl{valve\_p\_state\_cmd};

tank2\_ctrl\_e\_valve\_state : bool = t2\_ctrl{valve\_e\_state\_cmd};

Requirements:

r\_t1\_ctrl = global :: always t1\_ctrl == Ctrl\_Tanks.Tank1\_Ctrl(t1\_sensor\_high, t1\_sensor\_low);

r\_t2\_ctrl = global :: always t2\_ctrl == Ctrl\_Tanks.Tank2\_Ctrl(t2\_sensor\_high, t2\_sensor\_mid, t2\_sensor\_low, tank1\_ctrl\_valve\_state);

r\_output = global :: always ctrl\_output == new controller\_output\_type{pump\_state = tank1\_ctrl\_pump\_state,

t1\_valve\_state = tank1\_ctrl\_valve\_state,

t2\_p\_valve\_state = tank2\_ctrl\_p\_valve\_state,

t2\_e\_valve\_state = tank2\_ctrl\_e\_valve\_state};

Properties:

// tank 1 logic verification

p\_ctrl\_01 = while not t1\_sensor\_low :: always not tank1\_ctrl\_valve\_state;

p\_ctrl\_02 = while t1\_sensor\_high :: always not tank1\_ctrl\_pump\_state;

// tank 2 logic verification

p\_ctrl\_03 = while t2\_sensor\_low :: always tank2\_ctrl\_p\_valve\_state;

p\_ctrl\_04 = while t2\_sensor\_high :: always tank2\_ctrl\_p\_valve\_state and tank2\_ctrl\_e\_valve\_state;

canary = global :: always false;

==============================================================================

definitions.spear

==============================================================================

Definition Ctrl\_Tanks.definitions;

Units:

m; // meter

s; // second

m3 : m \* m \* m; // meters cubed

m3\_p\_s : m3 / s; // cubic meters per second

Constants:

// Tank 1 Constants ---------------------

TANK1\_SENSOR\_HIGH : real = 5.0;

TANK1\_SENSOR\_LOW : real = 3.0;

MAX\_PUMP\_FLOWRATE : real = 0.2;

MAX\_TANK1\_VALVE\_FLOWRATE : real = 0.1 ;

MAX\_TANK1\_HEIGHT : real = 7.0;

TANK1\_CROSS\_SECTIONAL\_AREA : real = 1.0;

TANK1\_INIT\_LIQUID\_HEIGHT : real = 0.0;

// --------------------------------------

// Tank 2 Constants ---------------------

TANK2\_SENSOR\_HIGH : real = 3.0;

TANK2\_SENSOR\_MID : real = 2.0;

TANK2\_SENSOR\_LOW : real = 1.0;

// Tank2's max inflow is Tank1's max outflow

MAX\_TANK2\_E\_VALVE\_FLOWRATE : real = 0.2;

MAX\_TANK2\_P\_VALVE\_FLOWRATE : real = 0.05;

MAX\_TANK2\_HEIGHT : real = 4.0;

TANK2\_CROSS\_SECTIONAL\_AREA : real = 2.0;

TANK2\_INIT\_LIQUID\_HEIGHT :real = 0.0;

// --------------------------------------

Types:

system\_output\_type : {production\_flow : real , emergency\_flow : real};

environment\_output\_type : {t1\_sensor\_high : bool, t1\_sensor\_low : bool,

t2\_sensor\_high : bool, t2\_sensor\_mid : bool, t2\_sensor\_low : bool,

tank1\_water\_lvl : real, tank1\_overflow : bool, tank1\_underflow : bool,

tank2\_water\_lvl : real, tank2\_overflow : bool, tank2\_underflow : bool,

production\_flow : real , emergency\_flow : real};

controller\_output\_type : {pump\_state : bool, t1\_valve\_state : bool, t2\_p\_valve\_state : bool, t2\_e\_valve\_state : bool};

// Tank output types

// tank1\_output\_type : {sensor\_high : bool, sensor\_low : bool, water\_lvl : real};

tank1\_output\_type : {water\_lvl : real, overflow : bool, underflow : bool};

// tank2\_output\_type : {sensor\_high : bool, sensor\_mid : bool, sensor\_low : bool, water\_lvl : real};

tank2\_output\_type : {water\_lvl : real, overflow : bool, underflow : bool};

// Tank controllers output types

tank1\_ctrl\_output\_type : {pump\_state\_cmd : bool, valve\_state\_cmd : bool};

tank2\_ctrl\_output\_type : {valve\_p\_state\_cmd : bool, valve\_e\_state\_cmd : bool};

==============================================================================

Environment.spear

==============================================================================

Relation Ctrl\_Tanks.Environment;

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.environment\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.tank1\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.tank2\_output\_type.\*;

Inputs:

t1\_pump\_state : bool;

t1\_valve\_state : bool;

t2\_prod\_valve\_state : bool;

t2\_emrg\_valve\_state : bool;

Outputs:

env\_output : environment\_output\_type;

State:

tank1\_output : tank1\_output\_type;

tank2\_output : tank2\_output\_type;

// get the outflows through the actuators in the system

available\_pump\_inflow : real;

pump\_outflow : real;

t1\_valve\_outflow : real;

t2\_p\_valve\_outflow : real;

t2\_e\_valve\_outflow : real;

Macros:

tank1\_output\_water\_lvl : real = tank1\_output{tank1\_output\_type.water\_lvl};

tank1\_overflow : bool = tank1\_output{tank1\_output\_type.overflow};

tank1\_underflow : bool = tank1\_output{tank1\_output\_type.underflow};

tank1\_sensor\_high : bool = Ctrl\_Tanks.Sensor(tank1\_output\_water\_lvl, TANK1\_SENSOR\_HIGH);

tank1\_sensor\_low : bool = Ctrl\_Tanks.Sensor(tank1\_output\_water\_lvl, TANK1\_SENSOR\_LOW);

tank2\_output\_water\_lvl : real = tank2\_output{tank2\_output\_type.water\_lvl};

tank2\_overflow : bool = tank2\_output{tank2\_output\_type.overflow};

tank2\_underflow : bool = tank2\_output{tank2\_output\_type.underflow};

tank2\_sensor\_high : bool = Ctrl\_Tanks.Sensor(tank2\_output\_water\_lvl, TANK2\_SENSOR\_HIGH);

tank2\_sensor\_mid : bool = Ctrl\_Tanks.Sensor(tank2\_output\_water\_lvl, TANK2\_SENSOR\_MID);

tank2\_sensor\_low : bool = Ctrl\_Tanks.Sensor(tank2\_output\_water\_lvl, TANK2\_SENSOR\_LOW);

// Making a simplification on the outflow that of each tank that it shouldn't drain unless it has enough liquid behind to drain

tank1\_available\_valve\_inflow : bool = tank1\_sensor\_low;

tank2\_available\_valve\_inflow : bool = tank2\_sensor\_low;

// t2\_p\_valve\_outflow : real = Ctrl\_Tanks.Valve(tank2\_available\_valve\_inflow, t2\_prod\_valve\_state, MAX\_TANK2\_P\_VALVE\_FLOWRATE);

Requirements:

a\_avail\_pump\_inflow = global :: always available\_pump\_inflow >= MAX\_PUMP\_FLOWRATE;

a\_p\_flow = initial :: t2\_p\_valve\_outflow == 0.0;

a\_e\_flow = initial :: t2\_e\_valve\_outflow == 0.0;

r\_t1\_pump = global :: always pump\_outflow == Ctrl\_Tanks.Pump(available\_pump\_inflow, t1\_pump\_state);

r\_t1\_valve = global :: always t1\_valve\_outflow == Ctrl\_Tanks.Valve(tank1\_available\_valve\_inflow, t1\_valve\_state, MAX\_TANK1\_VALVE\_FLOWRATE);

r\_t2\_p\_valve = global :: always t2\_p\_valve\_outflow == Ctrl\_Tanks.Valve(tank2\_available\_valve\_inflow, t2\_prod\_valve\_state, MAX\_TANK2\_P\_VALVE\_FLOWRATE);

r\_t2\_e\_valve = global :: always t2\_e\_valve\_outflow == Ctrl\_Tanks.Valve(tank2\_available\_valve\_inflow, t2\_emrg\_valve\_state, MAX\_TANK2\_E\_VALVE\_FLOWRATE);

r\_t1 = global :: always tank1\_output == Ctrl\_Tanks.Tank1(pump\_outflow,t1\_valve\_outflow);

r\_t2 = global :: always tank2\_output == Ctrl\_Tanks.Tank2(t1\_valve\_outflow,t2\_p\_valve\_outflow,t2\_e\_valve\_outflow);

r\_output = global :: always env\_output == new environment\_output\_type{t1\_sensor\_high = tank1\_sensor\_high,

t1\_sensor\_low = tank1\_sensor\_low,

t2\_sensor\_high = tank2\_sensor\_high,

t2\_sensor\_mid = tank2\_sensor\_mid,

t2\_sensor\_low = tank2\_sensor\_low,

tank1\_water\_lvl = tank1\_output\_water\_lvl,

tank1\_overflow = tank1\_overflow,

tank1\_underflow = tank1\_underflow,

tank2\_water\_lvl = tank2\_output\_water\_lvl,

tank2\_overflow = tank2\_overflow,

tank2\_underflow = tank2\_underflow,

production\_flow = t2\_p\_valve\_outflow,

emergency\_flow = t2\_e\_valve\_outflow };

Properties:

// Tank 1 sensor properties

p\_env\_01 = while tank1\_sensor\_high :: always tank1\_sensor\_low;

p\_env\_02 = while not tank1\_sensor\_low :: always not tank1\_sensor\_high;

// Tank 2 sensor properties

p\_env\_03 = while tank2\_sensor\_high :: always (tank2\_sensor\_mid and tank2\_sensor\_low);

p\_env\_04 = while tank2\_sensor\_mid :: always tank2\_sensor\_low;

p\_env\_05 = while not tank2\_sensor\_low :: always not tank2\_sensor\_mid and not tank2\_sensor\_high;

p\_env\_06 = while not tank2\_sensor\_mid :: always not tank2\_sensor\_high;

// Check the possible liquid levels in the tanks

p\_env\_07 = global :: always tank1\_output\_water\_lvl >= 0.0;

p\_env\_08 = global :: always tank1\_output\_water\_lvl <= MAX\_TANK1\_HEIGHT;

p\_env\_09 = global :: always tank2\_output\_water\_lvl >= 0.0;

p\_env\_10 = global :: always tank2\_output\_water\_lvl <= MAX\_TANK2\_HEIGHT;

// Counter-examples (designed to prove correct behavior of the system)

// Verify that the tanks can reach an overflow state (should be prevented by the controllers)

// p\_env\_CE01 = global :: always not tank1\_overflow;

// p\_env\_CE02 = global :: always not tank2\_overflow;

//

// // Verify that the tanks can reach an underflow state (should be prevented by the controllers)

// p\_env\_CE03 = global :: always not tank1\_underflow;

// p\_env\_CE04 = global :: always not tank2\_underflow;

//

// canary = global :: always false;

==============================================================================

project.xml

==============================================================================

<?xml version="1.0" encoding="UTF-8"?>

<projectDescription>

<name>control\_tanks</name>

<comment></comment>

<projects>

</projects>

<buildSpec>

<buildCommand>

<name>org.eclipse.xtext.ui.shared.xtextBuilder</name>

<arguments>

</arguments>

</buildCommand>

</buildSpec>

<natures>

<nature>org.eclipse.xtext.ui.shared.xtextNature</nature>

</natures>

</projectDescription>

==============================================================================

Pump.spear

==============================================================================

Relation Ctrl\_Tanks.Pump;

Uses Ctrl\_Tanks.definitions.\*;

Inputs:

inflow : real;

pump\_on\_state : bool;

Outputs:

outflow : real;

State:

Requirements:

a01 = global :: always inflow >= MAX\_PUMP\_FLOWRATE;

r01 = while pump\_on\_state :: always outflow == MAX\_PUMP\_FLOWRATE;

r02 = while not pump\_on\_state :: always outflow == 0.0 ;

Properties:

p\_pump\_01 = global :: always outflow >= 0.0;

// canary = global :: always false;

==============================================================================

Pump.spear.lus

==============================================================================

node initial(a : bool) returns (holds : bool);

let

holds = a -> true ;

tel ;

node before\_always(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 1

else if(not p and a)

then 0

else if(not p and not a)

then 2

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then

if(not p)

then 2

else if p

then 3

else -1

else if(pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

tel;

node before\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 2

else if(not p and not a)

then 0

else if(not p and a)

then 1

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node before\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 1

else if a

then 1

else if (not p and not a and b)

then 2

else if (not p and not a and not b)

then 0

else -1

else if (pre\_state = 1)

then 1

else if (pre\_state = 2)

then

if p

then 3

else if (not p)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node before\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 2

else if (not p and not b)

then 0

else if (not p and a and b)

then 0

else if (not p and not a and b)

then 1

else -1

else if (pre\_state = 1)

then

if p

then 3

else if (not p and not a)

then 1

else if (not p and a)

then 0

else -1

else if (pre\_state = 2)

then 2

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if (not a)

then 1

else if a

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state = 0) or (state = 2);

tel;

node after\_always(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then 1

else if (pre\_state = 2)

then

if a then 2

else 1

else -1;

holds = (state = 0) or (state = 2);

tel;

node after\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 3

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 2

else if (not a and not b)

then 1

else if (not a and b)

then 3

else -1

else if (pre\_state = 2)

then 2

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 1

else if (not a and not b)

then 1

else if (not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if a

then 1

else if (not a)

then 2

else -1

else -1;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_always (p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 0

else if (p and not a)

then 1

else -1

else if(pre\_state = 1)

then 1

else -1;

holds = (state <> 1);

tel;

node while\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 3

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 2)

then

if (not p)

then 0

else if p

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state,pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if(not p)

then 0

else if (p and a)

then 0

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 0

else if a

then 0

else if(p and not a and not b)

then 1

else if(p and not a and b)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if (not p)

then 3

else if (p and a)

then 1

else if (p and not a)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1 ;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_always(a : bool) returns (holds : bool );

let

holds = a and (true -> pre holds);

tel;

node global\_responds\_within(a : bool; b : bool; n : int) returns (holds : bool);

var

state,pre\_state : subrange [-1,2] of int;

latch\_n : int;

i,pre\_i : int;

let

latch\_n = n -> pre latch\_n;

pre\_i = 0 -> pre i;

i =

if (pre\_state = 0)

then

if (b and not a)

then 0

else pre\_i

else if (pre\_state = 1)

then

if (not a)

then pre\_i + 1

else pre\_i

else if (pre\_state = 2)

then pre\_i

else pre\_i;

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not b)

then 0

else if(b and a)

then 0

else if(b and not a and latch\_n > 0)

then 1

else if(b and not a and latch\_n = 0)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 0

else if (not a and i < latch\_n)

then 1

else if (not a and i >= latch\_n)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state = 0) or (state = 1);

tel;

node global\_exists(a : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not a)

then 0

else if a

then 1

else -1

else if (pre\_state = 1)

then 1

else -1;

holds = (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_precedes (a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if a

then 1

else if(not a and not b)

then 0

else if(not a and b)

then 2

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_responds(a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not b)

then 0

else if (a and b)

then 0

else if (not a and b)

then 1

else -1

else if (pre\_state = 1)

then

if (not a)

then 1

else if a

then 0

else -1

else -1;

holds = (state = 0);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_always (p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if(not q and a)

then 1

else if(not q and not a)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

tel;

node between\_exists(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if q

then 0

else if a

then 0

else if (p and not q and not a)

then 1

else -1

else if (pre\_state = 1)

then

if q

then 2

else if (not q and not a)

then 1

else if (not q and a)

then 0

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_precedes (p : bool; q : bool; a : bool ; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if not p

then 0

else if (p and q)

then 0

else if (p and a)

then 0

else if (p and not q and not a and not b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if(pre\_state = 1)

then

if q

then 0

else if a

then 0

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if(pre\_state = 2)

then

if q

then 3

else if (not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_responds (p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and q)

then 0

else if (p and not q and not b)

then 1

else if (p and not q and a and b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if (not q and not b)

then 1

else if (not q and a and b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q and not a)

then 2

else if (not q and a)

then 1

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_always(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a)

then 2

else -1

else if(pre\_state = 1)

then

if q

then 0

else if (not q and a)

then 1

else if (not q and not a)

then 2

else -1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

tel;

node after\_until\_exists(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if(p and q)

then 0

else if(p and not a and not q)

then 1

else if(p and a and not q)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 3

else if (not q and a)

then 2

else if (not q and not a)

then 1

else -1

else if (pre\_state = 2)

then

if q

then 0

else if (p and not q and a)

then 2

else if (p and not q and not a)

then 1

else if (not p and not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state = 0) or (state = 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_precedes(p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and q)

then 0

else if (p and not q and a)

then 0

else if (p and not q and not a and not b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if a

then 0

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_responds(p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a and not b)

then 1

else if(p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if (not q and a)

then 1

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q and not a)

then 2

else if (not q and a)

then 1

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node H(a : bool) returns (holds : bool);

let

holds = a and Z(holds);

tel;

node O(a : bool) returns (holds : bool);

let

holds = a or Y(holds);

tel;

node Y(a : bool) returns (holds : bool);

let

holds = (false -> pre a);

tel;

node Z(a : bool) returns (holds : bool);

let

holds = (true -> pre a);

tel;

node S(a : bool; b : bool) returns (holds : bool);

let

holds = b or (a and Y(holds));

tel;

node T(a : bool; b : bool) returns (holds : bool);

let

holds = b and (a or Z(holds));

tel;

node holds\_while(p : bool; c : bool) returns (holds : bool);

let

holds = (Y(p) and c) => p;

tel;

const Ctrl\_Tanks\_definitions\_MAX\_PUMP\_FLOWRATE = 0.2;

node Ctrl\_Tanks\_Pump(

inflow : real;

pump\_on\_state : bool;

outflow : real

) returns (

\_gen\_requirements\_0 : bool

);

var

a01 : bool;

r01 : bool;

r02 : bool;

p01 : bool;

canary : bool;

let

--%MAIN

a01 = global\_always((inflow >= 0.2));

r01 = while\_always(pump\_on\_state, (outflow = 0.2));

r02 = while\_always((not pump\_on\_state), (outflow = 0.0));

p01 = (\_gen\_requirements\_0 => global\_always((outflow >= 0.0)));

canary = (\_gen\_requirements\_0 => global\_always(false));

\_gen\_requirements\_0 = H(((a01 and r01) and r02));

--%PROPERTY p01;

--%PROPERTY canary;

tel;

==============================================================================

Sensor.spear

==============================================================================

Relation Ctrl\_Tanks.Sensor;

// This is the requirements on binary liquid height sensors

// -> Once the liquid reaches the placement height of the sensor it triggers an ON (ACTIVE) state

// -> If the water has not reached the sensor it does not produce a signal (FALSE/INACTIVE)

Uses Ctrl\_Tanks.definitions.\*;

Inputs:

liquid\_height : real;

placement\_height : real;

Outputs:

active : bool;

State:

Requirements:

// assumptions on the inputs

a\_liq\_ht = global :: always liquid\_height >= 0.0;

a\_pla\_ht = global :: always placement\_height >= 0.0;

r\_active = while liquid\_height >= placement\_height :: always active;

r\_inactive = while liquid\_height < placement\_height :: always not active;

Properties:

p\_sensor\_01 = while active :: always liquid\_height >= placement\_height;

p\_sensor\_02 = while not active :: always liquid\_height < placement\_height;

// These properties will prove due to the assumptions but is not under the control of this component

// p\_sensor\_03 = global :: always placement\_height >= 0.0;

// p\_sensor\_04 = global :: always liquid\_height >= 0.0;

// canary = global :: always false;

==============================================================================

Sensor.spear.lus

==============================================================================

node initial(a : bool) returns (holds : bool);

let

holds = a -> true ;

tel ;

node before\_always(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 1

else if(not p and a)

then 0

else if(not p and not a)

then 2

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then

if(not p)

then 2

else if p

then 3

else -1

else if(pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

tel;

node before\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 2

else if(not p and not a)

then 0

else if(not p and a)

then 1

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node before\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 1

else if a

then 1

else if (not p and not a and b)

then 2

else if (not p and not a and not b)

then 0

else -1

else if (pre\_state = 1)

then 1

else if (pre\_state = 2)

then

if p

then 3

else if (not p)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node before\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if p

then 2

else if (not p and not b)

then 0

else if (not p and a and b)

then 0

else if (not p and not a and b)

then 1

else -1

else if (pre\_state = 1)

then

if p

then 3

else if (not p and not a)

then 1

else if (not p and a)

then 0

else -1

else if (pre\_state = 2)

then 2

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if (not a)

then 1

else if a

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state = 0) or (state = 2);

tel;

node after\_always(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then 1

else if (pre\_state = 2)

then

if a then 2

else 1

else -1;

holds = (state = 0) or (state = 2);

tel;

node after\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 3

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 2

else if (not a and not b)

then 1

else if (not a and b)

then 3

else -1

else if (pre\_state = 2)

then 2

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 1

else if (not a and not b)

then 1

else if (not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if a

then 1

else if (not a)

then 2

else -1

else -1;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_always (p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 0

else if (p and not a)

then 1

else -1

else if(pre\_state = 1)

then 1

else -1;

holds = (state <> 1);

tel;

node while\_exists(p : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 3

else if (p and not a)

then 1

else if (p and a)

then 2

else -1

else if (pre\_state = 2)

then

if (not p)

then 0

else if p

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_precedes(p : bool; a : bool; b : bool) returns (holds : bool);

var

state,pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if(not p)

then 0

else if (p and a)

then 0

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 0

else if a

then 0

else if(p and not a and not b)

then 1

else if(p and not a and b)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node while\_responds(p : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if (not p)

then 0

else if (p and a)

then 1

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if (not p)

then 3

else if (p and a)

then 1

else if (p and not a)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1 ;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_always(a : bool) returns (holds : bool );

let

holds = a and (true -> pre holds);

tel;

node global\_responds\_within(a : bool; b : bool; n : int) returns (holds : bool);

var

state,pre\_state : subrange [-1,2] of int;

latch\_n : int;

i,pre\_i : int;

let

latch\_n = n -> pre latch\_n;

pre\_i = 0 -> pre i;

i =

if (pre\_state = 0)

then

if (b and not a)

then 0

else pre\_i

else if (pre\_state = 1)

then

if (not a)

then pre\_i + 1

else pre\_i

else if (pre\_state = 2)

then pre\_i

else pre\_i;

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not b)

then 0

else if(b and a)

then 0

else if(b and not a and latch\_n > 0)

then 1

else if(b and not a and latch\_n = 0)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 0

else if (not a and i < latch\_n)

then 1

else if (not a and i >= latch\_n)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state = 0) or (state = 1);

tel;

node global\_exists(a : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not a)

then 0

else if a

then 1

else -1

else if (pre\_state = 1)

then 1

else -1;

holds = (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_precedes (a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if a

then 1

else if(not a and not b)

then 0

else if(not a and b)

then 2

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node global\_responds(a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,1] of int;

pre\_state : subrange [-1,1] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not b)

then 0

else if (a and b)

then 0

else if (not a and b)

then 1

else -1

else if (pre\_state = 1)

then

if (not a)

then 1

else if a

then 0

else -1

else -1;

holds = (state = 0);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_always (p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if(not q and a)

then 1

else if(not q and not a)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

tel;

node between\_exists(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if q

then 0

else if a

then 0

else if (p and not q and not a)

then 1

else -1

else if (pre\_state = 1)

then

if q

then 2

else if (not q and not a)

then 1

else if (not q and a)

then 0

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_precedes (p : bool; q : bool; a : bool ; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if(pre\_state = 0)

then

if not p

then 0

else if (p and q)

then 0

else if (p and a)

then 0

else if (p and not q and not a and not b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if(pre\_state = 1)

then

if q

then 0

else if a

then 0

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if(pre\_state = 2)

then

if q

then 3

else if (not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node between\_responds (p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and q)

then 0

else if (p and not q and not b)

then 1

else if (p and not q and a and b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if (not q and not b)

then 1

else if (not q and a and b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q and not a)

then 2

else if (not q and a)

then 1

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state <> 3);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_always(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a)

then 2

else -1

else if(pre\_state = 1)

then

if q

then 0

else if (not q and a)

then 1

else if (not q and not a)

then 2

else -1

else if(pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

tel;

node after\_until\_exists(p : bool; q : bool; a : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if(p and q)

then 0

else if(p and not a and not q)

then 1

else if(p and a and not q)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 3

else if (not q and a)

then 2

else if (not q and not a)

then 1

else -1

else if (pre\_state = 2)

then

if q

then 0

else if (p and not q and a)

then 2

else if (p and not q and not a)

then 1

else if (not p and not q)

then 2

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state = 0) or (state = 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_precedes(p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,2] of int;

pre\_state : subrange [-1,2] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and q)

then 0

else if (p and not q and a)

then 0

else if (p and not q and not a and not b)

then 1

else if (p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if a

then 0

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node after\_until\_responds(p : bool; q : bool; a : bool; b : bool) returns (holds : bool);

var

state : subrange [-1,3] of int;

pre\_state : subrange [-1,3] of int;

prop1 : bool;

let

pre\_state = 0 -> pre state;

state =

if (pre\_state = 0)

then

if(not p)

then 0

else if(p and q)

then 0

else if(p and not q and a)

then 1

else if(p and not q and not a and not b)

then 1

else if(p and not q and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if q

then 0

else if (not q and a)

then 1

else if (not q and not a and not b)

then 1

else if (not q and not a and b)

then 2

else -1

else if (pre\_state = 2)

then

if q

then 3

else if (not q and not a)

then 2

else if (not q and a)

then 1

else -1

else if (pre\_state = 3)

then 3

else -1;

holds = (state = 0) or (state = 1);

prop1 = (state <> -1);

--%PROPERTY prop1;

tel;

node H(a : bool) returns (holds : bool);

let

holds = a and Z(holds);

tel;

node O(a : bool) returns (holds : bool);

let

holds = a or Y(holds);

tel;

node Y(a : bool) returns (holds : bool);

let

holds = (false -> pre a);

tel;

node Z(a : bool) returns (holds : bool);

let

holds = (true -> pre a);

tel;

node S(a : bool; b : bool) returns (holds : bool);

let

holds = b or (a and Y(holds));

tel;

node T(a : bool; b : bool) returns (holds : bool);

let

holds = b and (a or Z(holds));

tel;

node holds\_while(p : bool; c : bool) returns (holds : bool);

let

holds = (Y(p) and c) => p;

tel;

node Ctrl\_Tanks\_Sensor(

liquid\_height : real;

placement\_height : real;

active : bool

) returns (

\_gen\_requirements\_0 : bool

);

var

r\_active : bool;

r\_inactive : bool;

p\_sensor\_01 : bool;

p\_sensor\_02 : bool;

p\_sensor\_03 : bool;

let

--%MAIN

r\_active = while\_always((liquid\_height >= placement\_height), active);

r\_inactive = while\_always((liquid\_height < placement\_height), (not active));

p\_sensor\_01 = (\_gen\_requirements\_0 => global\_always((placement\_height >= 0.0)));

p\_sensor\_02 = (\_gen\_requirements\_0 => while\_always(active, (liquid\_height >= placement\_height)));

p\_sensor\_03 = (\_gen\_requirements\_0 => while\_always((not active), (liquid\_height < placement\_height)));

\_gen\_requirements\_0 = H((r\_active and r\_inactive));

--%PROPERTY p\_sensor\_01;

--%PROPERTY p\_sensor\_02;

--%PROPERTY p\_sensor\_03;

tel;

==============================================================================

SystemLevel.spear

==============================================================================

Relation Ctrl\_Tanks.SystemLevel;

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.system\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.controller\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.environment\_output\_type.\*;

Uses Ctrl\_Tanks.definitions.system\_output\_type.\*;

Inputs:

available\_inflow : real;

Outputs:

system\_outflow : system\_output\_type;

State:

env\_output : environment\_output\_type;

ctrl\_output : controller\_output\_type;

Macros:

// Commands for the environment from the controller

t1\_pump\_state : bool = ctrl\_output{controller\_output\_type.pump\_state};

t1\_valve\_state : bool = ctrl\_output{controller\_output\_type.t1\_valve\_state};

t2\_prod\_valve\_state : bool = ctrl\_output{controller\_output\_type.t2\_p\_valve\_state};

t2\_emrg\_valve\_state : bool = ctrl\_output{controller\_output\_type.t2\_e\_valve\_state};

// Signals for the controller from the environment

t1\_sensor\_high : bool = env\_output{environment\_output\_type.t1\_sensor\_high};

t1\_sensor\_low : bool = env\_output{environment\_output\_type.t1\_sensor\_low};

t2\_sensor\_high : bool = env\_output{environment\_output\_type.t2\_sensor\_high};

t2\_sensor\_mid : bool = env\_output{environment\_output\_type.t2\_sensor\_mid};

t2\_sensor\_low : bool = env\_output{environment\_output\_type.t2\_sensor\_low};

// Tank information from the environment

tank1\_water\_lvl : real = env\_output{environment\_output\_type.tank1\_water\_lvl};

tank1\_overflow : bool = env\_output{environment\_output\_type.tank1\_overflow};

tank1\_underflow : bool = env\_output{environment\_output\_type.tank1\_underflow};

tank2\_water\_lvl : real = env\_output{environment\_output\_type.tank2\_water\_lvl};

tank2\_overflow : bool = env\_output{environment\_output\_type.tank2\_overflow};

tank2\_underflow : bool = env\_output{environment\_output\_type.tank2\_underflow};

// Output flows from the system

system\_production\_flow : real = env\_output{environment\_output\_type.production\_flow};

system\_emergency\_flow : real = env\_output{environment\_output\_type.emergency\_flow};

Requirements:

a00 = global :: always available\_inflow >= MAX\_PUMP\_FLOWRATE;

a01 = initial :: system\_outflow{system\_output\_type.production\_flow} == 0.0;

a02 = initial :: system\_outflow{system\_output\_type.emergency\_flow} == 0.0;

r00 = global :: always env\_output == Ctrl\_Tanks.Environment(t1\_pump\_state, t1\_valve\_state, t2\_prod\_valve\_state, t2\_emrg\_valve\_state);

r01 = global :: always ctrl\_output == Ctrl\_Tanks.Controller(t1\_sensor\_high, t1\_sensor\_low, t2\_sensor\_high, t2\_sensor\_mid, t2\_sensor\_low);

r\_output = global :: always system\_outflow == new system\_output\_type{system\_output\_type.production\_flow = system\_production\_flow,

system\_output\_type.emergency\_flow = system\_emergency\_flow};

Properties:

// Check the possible liquid levels in the tanks

p\_sys\_01 = global :: always tank1\_water\_lvl >= 0.0;

p\_sys\_02 = global :: always tank1\_water\_lvl <= MAX\_TANK1\_HEIGHT;

p\_sys\_03 = global :: always tank2\_water\_lvl >= 0.0;

p\_sys\_04 = global :: always tank2\_water\_lvl <= MAX\_TANK2\_HEIGHT;

// Verify that the controllers protect the tanks from overflow or underflow

p\_sys\_05 = global :: always not tank1\_overflow;

p\_sys\_06 = global :: always not tank1\_underflow;

p\_sys\_07 = global :: always not tank2\_overflow;

p\_sys\_08 = global :: always not tank2\_underflow;

// System flow should never be negative (backflow)

p\_sys\_09 = global :: always system\_production\_flow >= 0.0;

p\_sys\_10 = global :: always system\_emergency\_flow >= 0.0;

canary = global :: always false;

==============================================================================

Tank1.spear

==============================================================================

Relation Ctrl\_Tanks.Tank1;

// This is the requirements on first tank (tank1) subcomponent

// -> This is designed to act only as a liquid container, any logic to prevent overflow and underflow

// is not the responsibility of this component.

// -> In the event that an overflow or underflow occurs the component must inform the rest of the system.

// -> This is an OPEN tank, therefore in the event the tank overflows then extra liquid will exceed the capacity

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.tank1\_output\_type.\*;

Inputs:

inflow : real; // dependent on the current pump flow rate

outflow : real; // dependent on the current valve flow rate

Outputs:

output : tank1\_output\_type; // only output information that the tank can provide

State:

level : real;

overflow\_state : bool;

underflow\_state : bool;

Macros:

calculated\_height : real = (pre level + ((inflow - outflow) / TANK1\_CROSS\_SECTIONAL\_AREA));

tank\_liquid\_height : real = if (calculated\_height < 0.0) then 0.0 else

if (calculated\_height > MAX\_TANK1\_HEIGHT) then MAX\_TANK1\_HEIGHT else

calculated\_height;

Requirements:

a01 = initial :: level == TANK1\_INIT\_LIQUID\_HEIGHT;

a\_inflow\_1 = global :: always inflow >= 0.0;

a\_inflow\_2 = global :: always inflow <= MAX\_PUMP\_FLOWRATE;

a\_outflow\_1 = global :: always outflow >= 0.0;

a\_outflow\_2 = global :: always outflow <= MAX\_TANK1\_VALVE\_FLOWRATE;

r\_liq\_ht = global :: always level == tank\_liquid\_height;

// Not currently working - Lucas is looking into

// r\_overflow\_00 = before level >= MAX\_TANK1\_HEIGHT :: always not overflow\_state;

// r\_overflow\_01 = after level >= MAX\_TANK1\_HEIGHT :: always overflow\_state;

// This works in the trace because we are using the gross height calculation to determine underflow and overflow

// -> The trace is only showing the level which is protected to the proper height restrictions

r\_overflow\_00 = while calculated\_height <= MAX\_TANK1\_HEIGHT :: always not overflow\_state;

r\_overflow\_01 = while calculated\_height > MAX\_TANK1\_HEIGHT :: always overflow\_state;

r\_underflow\_00 = while calculated\_height >= 0.0 :: always not underflow\_state;

r\_underflow\_01 = while calculated\_height < 0.0 :: always underflow\_state;

r\_output = global :: always output == new tank1\_output\_type{tank1\_output\_type.water\_lvl = level,

tank1\_output\_type.overflow = overflow\_state,

tank1\_output\_type.underflow = underflow\_state};

Properties:

// Verify liquid height remains in correct range

p\_t1\_01 = global :: always level >= 0.0;

p\_t1\_02 = global :: always level <= MAX\_TANK1\_HEIGHT; // the tank can't hold more than the max height

// Counter-examples (designed to prove correct behavior of the system)

// p\_t1\_CE01 = global :: always not overflow\_state;

// p\_t1\_CE02 = global :: always not underflow\_state;

// canary = global :: always false;

==============================================================================

Tank1\_Ctrl.spear

==============================================================================

Relation Ctrl\_Tanks.Tank1\_Ctrl;

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.tank1\_ctrl\_output\_type.\*;

Inputs:

sensor\_high : bool;

sensor\_low : bool;

Outputs:

output : tank1\_ctrl\_output\_type;

State:

pump\_state : bool;

valve\_state : bool;

Requirements:

a01 = global :: always sensor\_high implies sensor\_low;

a02 = global :: always not sensor\_low implies not sensor\_high;

init = initial :: pump\_state and not valve\_state;

r00 = while sensor\_high :: always not pump\_state and valve\_state;

r01 = while not sensor\_low :: always pump\_state and not valve\_state;

r02 = while not sensor\_high and sensor\_low :: always (pump\_state == pre pump\_state) and (valve\_state == pre valve\_state);

r\_output = global :: always output == new tank1\_ctrl\_output\_type{pump\_state\_cmd = pump\_state, valve\_state\_cmd = valve\_state};

Properties:

p01 = global :: always not (pump\_state and valve\_state);

// canary = global :: always false;

==============================================================================

Tank2.spear

==============================================================================

Relation Ctrl\_Tanks.Tank2;

// This is the requirements on second tank (tank2) subcomponent

// -> This is designed to act only as a liquid container, any logic to prevent overflow and underflow

// is not the responsibility of this component.

// -> In the event that an overflow or underflow occurs the component must inform the rest of the system.

// -> This is an OPEN tank, therefore in the event the tank overflows then extra liquid will exceed the capacity

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.tank2\_output\_type.\*;

Inputs:

inflow : real; // dependent on the current valve tank1 outflow valve

production\_outflow : real; // dependent on the current production outflow valve

emergency\_outflow : real; // dependent on the current emergency outflow valve

Outputs:

output : tank2\_output\_type;

State:

level : real;

overflow\_state : bool;

underflow\_state : bool;

Macros:

total\_outflow : real = production\_outflow + emergency\_outflow;

calculated\_height : real = (pre level + ((inflow - total\_outflow) / TANK2\_CROSS\_SECTIONAL\_AREA));

tank\_liquid\_height : real = if (calculated\_height < 0.0) then 0.0 else

if (calculated\_height > MAX\_TANK2\_HEIGHT) then MAX\_TANK2\_HEIGHT else

calculated\_height;

// tank\_overflow : bool = if (calculated\_height > MAX\_TANK2\_HEIGHT) then true else false;

Requirements:

a01 = initial :: level == TANK2\_INIT\_LIQUID\_HEIGHT;

a\_inflow\_1 = global :: always inflow >= 0.0;

a\_inflow\_2 = global :: always inflow <= MAX\_TANK1\_VALVE\_FLOWRATE; // could argue that this is a property of the valve and not appropriate here

a\_p\_outflow\_1 = global :: always production\_outflow >= 0.0;

a\_p\_outflow\_2 = global :: always production\_outflow <= MAX\_TANK2\_P\_VALVE\_FLOWRATE;

a\_e\_outflow\_1 = global :: always emergency\_outflow >= 0.0;

a\_e\_outflow\_2 = global :: always emergency\_outflow <= MAX\_TANK2\_E\_VALVE\_FLOWRATE;

r\_liq\_ht = global :: always level == tank\_liquid\_height;

// Not currently working - Lucas is looking into

// r\_overflow\_00 = before tank\_overflow :: always not overflow\_state;

// r\_overflow\_01 = after tank\_overflow :: always overflow\_state;

// This works in the trace because we are using the gross height calculation to determine underflow and overflow

// -> The trace is only showing the level which is protected to the proper height restrictions

r\_overflow\_00 = while calculated\_height <= MAX\_TANK2\_HEIGHT :: always not overflow\_state;

r\_overflow\_01 = while calculated\_height > MAX\_TANK2\_HEIGHT :: always overflow\_state;

r\_underflow\_00 = while calculated\_height >= 0.0 :: always not underflow\_state;

r\_underflow\_01 = while calculated\_height < 0.0 :: always underflow\_state;

r\_output = global :: always output == new tank2\_output\_type {tank2\_output\_type.water\_lvl = level,

tank2\_output\_type.overflow = overflow\_state,

tank2\_output\_type.underflow = underflow\_state};

Properties:

// Verify liquid height remains in correct range

p\_t2\_01 = global :: always level >= 0.0;

p\_t2\_02 = global :: always level <= MAX\_TANK2\_HEIGHT; // the tank can't hold more than the max height

p\_t2\_03 = global :: always total\_outflow == (production\_outflow + emergency\_outflow);

// Counter-examples (designed to prove correct behavior of the system)

// p\_t2\_CE01 = global :: always not overflow\_state;

// p\_t2\_CE02 = global :: always not underflow\_state;

// canary = global :: always false;

==============================================================================

Tank2\_Ctrl.spear

==============================================================================

Relation Ctrl\_Tanks.Tank2\_Ctrl;

Uses Ctrl\_Tanks.definitions.\*;

Uses Ctrl\_Tanks.definitions.tank2\_ctrl\_output\_type.\*;

Inputs:

sensor\_high : bool;

sensor\_mid : bool;

sensor\_low : bool;

input\_valve\_state : bool;

State:

e\_valve : bool;

p\_valve : bool;

Outputs:

output : tank2\_ctrl\_output\_type;

Requirements:

a01 = global :: always sensor\_high implies (sensor\_mid and sensor\_low);

a02 = global :: always sensor\_mid implies sensor\_low;

a03 = global :: always not sensor\_low implies not (sensor\_mid or sensor\_high);

a04 = global :: always not sensor\_mid implies not sensor\_high;

r00 = initial :: not (p\_valve and e\_valve);

r01 = while not sensor\_low :: always not (e\_valve or p\_valve);

r02 = while sensor\_low and not sensor\_mid :: always not e\_valve and p\_valve;

r03 = while sensor\_mid and not sensor\_high :: always (e\_valve == input\_valve\_state) and p\_valve;

r04 = while sensor\_high :: always e\_valve and p\_valve;

r\_output = global :: always output == new tank2\_ctrl\_output\_type{valve\_p\_state\_cmd = p\_valve, valve\_e\_state\_cmd = e\_valve};

Properties:

p\_t2ctrl\_01 = after sensor\_high until (not sensor\_mid or not input\_valve\_state) :: always e\_valve;

p\_t2ctrl\_02 = after sensor\_low until not sensor\_low :: always p\_valve;

// canary = global :: always false;

==============================================================================

Valve.spear

==============================================================================

Relation Ctrl\_Tanks.Valve;

Uses Ctrl\_Tanks.definitions.\*;

Inputs:

available\_inflow : bool; // basically the tank will provide true if the water behind the tank is of acceptable level for the valve to be open

valve\_open\_state : bool;

max\_flowrate : real;

Outputs:

outflow : real;

State:

Macros:

// differential : real = max\_flowrate - available\_inflow;

Requirements:

// a00 = global :: always differential == 0.0;

// Make an assumption here about the available inflow, if the low sensor in the tank is on then assume the necessary liquid is available

// -> this gets around doing an mathematical calculation and acts more like our system

// -> also gets around the physics of the flow

a\_inflow\_1 = while not available\_inflow :: always outflow == 0.0;

a\_inflow\_2 = while not available\_inflow :: always not valve\_open\_state;

r01 = while valve\_open\_state :: always outflow > 0.0 and outflow <= max\_flowrate;

r02 = while not valve\_open\_state :: always outflow == 0.0 ;

Properties:

p\_valve\_01 = global :: always outflow >= 0.0;

p\_valve\_02 = while valve\_open\_state :: always available\_inflow;

// canary = global :: always false;

# Appendix C

This is the source code for the AGREE functional representation of the Basic Control Tanks System

==============================================================================

.aadlsettings

==============================================================================

#Thu Apr 17 09:09:58 EDT 2014

source.directory=/aadl

model.directory=/aaxl

==============================================================================

.project

==============================================================================

<?xml version="1.0" encoding="UTF-8"?>

<projectDescription>

<name>Ctrl\_Tanks</name>

<comment></comment>

<projects>

<project>Plugin\_Resources</project>

</projects>

<buildSpec>

<buildCommand>

<name>org.eclipse.xtext.ui.shared.xtextBuilder</name>

<arguments>

</arguments>

</buildCommand>

</buildSpec>

<natures>

<nature>org.eclipse.xtext.ui.shared.xtextNature</nature>

<nature>org.osate.core.aadlnature</nature>

</natures>

</projectDescription>

==============================================================================

config.aadl

==============================================================================

package config

public

annex agree {\*\*

-- time rates

const system\_rate\_s : int = 1;

const init\_period\_s : int = system\_rate\_s \* 20;

const high\_safety\_margin\_s : int = system\_rate\_s \* 3;

const low\_safety\_margin\_s : int = system\_rate\_s \* 3;

-- Tank 1 specifications ----------------------------

-- volume for Tank 1

const max\_tank1\_vol\_m3 : real = 7.0;

const tank1\_empty\_m3 : real = 0.0;

const tank1\_initial\_vol\_m3 : real = 0.0;

-- parameters for Tank 1

const max\_tank1\_height\_m : real = 7.0;

const tank1\_empty\_m : real = 0.0;

const tank1\_lo\_height\_m : real = 2.0;

const tank1\_hi\_height\_m : real = 5.0;

const tank1\_cross\_section\_area\_m2 : real = 1.0;

-- safety parameters for Tank 1

const tank1\_safety\_hi\_ht\_m : real = 5.0;

const tank1\_safety\_lo\_ht\_m : real = 2.0;

-- sensor heights

const t1\_sensor\_lo\_height\_m : real = 2.0;

const t1\_sensor\_hi\_height\_m : real = 5.0;

-- valve const

const t1\_valve\_max\_flow\_rate\_m3s : real = 0.1;

const t1\_valve\_zero\_flow\_rate\_m3s : real = 0.0;

-- pump const

const t1\_pump\_max\_flow\_rate\_m3s : real = 0.2;

const t1\_pump\_zero\_flow\_rate\_m3s : real = 0.0;

-----------------------------------------------------

-- Tank 2 specifications ----------------------------

-- volume for Tank 2

const max\_tank2\_vol\_m3 : real = 8.0;

const tank2\_empty\_m3 : real = 0.0;

const tank2\_initial\_vol\_m3 : real = 0.0;

-- parameters for Tank 2

const max\_tank2\_height\_m : real = 4.0;

const tank2\_empty\_m : real = 0.0;

const tank2\_lo\_height\_m : real = 1.0;

const tank2\_md\_height\_m : real = 2.0;

const tank2\_hi\_height\_m : real = 3.0;

const tank2\_cs\_area\_m2 : real = 2.0;

-- safety parameters for Tank 2

const tank2\_safety\_hi\_ht\_m : real = 3.0;

const tank2\_safety\_md\_ht\_m : real = 2.0;

const tank2\_safety\_lo\_ht\_m : real = 1.0;

-- sensor heights

const t2\_sensor\_hi\_height\_m : real = 3.0;

const t2\_sensor\_md\_height\_m : real = 2.0;

const t2\_sensor\_lo\_height\_m : real = 1.0;

-- valve const

const t2\_e\_valve\_max\_flow\_rate\_m3s : real = 0.2;

const t2\_p\_valve\_max\_flow\_rate\_m3s : real = 0.05;

const t2\_valve\_zero\_flow\_rate\_m3s : real = 0.0;

-----------------------------------------------------

\*\*};

end config;

==============================================================================

controller.aadl

==============================================================================

package controller\_package

public

with Base\_Types;

with std;

with valve\_package;

with pump\_package;

with sensor\_package;

system Controller\_Tank1

features

SH\_Input: in data port Base\_Types::Boolean;

SL\_Input: in data port Base\_Types::Boolean;

Pump\_State: out data port Base\_Types::Boolean; -- commanded pump state and valve state

Valve\_State: out data port Base\_Types::Boolean;

annex agree {\*\*

-- assumption: goal is to keep the fluid between the sensors

-- Initial state: assume pump and valve are off

-- assume "The pump is initially off":

guarantee "The pump is initially off":

std.Init(not Pump\_State);

guarantee "The valve is initially closed" :

std.Init(not Valve\_State);

-- Liquid below Below Low sensor - just flow in

guarantee "When SL\_Input is False, the Pump shall be on and Valve shall be Closed" :

true -> ((not SL\_Input) => ((not Valve\_State) and Pump\_State));

-- Liquid above high sensor - just flow out

guarantee "When the SH\_Input is true, the Pump shall be off and Valve shall be open" :

true -> (SH\_Input => (Valve\_State and (not Pump\_State)));

-- Liquid between sensors - hold pump and valve states

guarantee "When the SL\_Input is True and the SH\_Input is False, the Pump and Valve shall mainatin the state they were in the previous timestep" :

true -> ((not SH\_Input and SL\_Input) => (Valve\_State = pre(Valve\_State) and Pump\_State = pre(Pump\_State)));

-- System description

assume "If the high sensor is true, then the low sensor is True" :

(SH\_Input => SL\_Input);

\*\*};

end Controller\_Tank1;

system implementation Controller\_Tank1.Impl

annex agree {\*\*

-- when to turn pump on - as long as liquid isn't at high sensor

eq pump\_command : bool = false ->

(

if (not SH\_Input and not SL\_Input) then true

else if (SH\_Input and SL\_Input) then false

else (pump\_command = pre(pump\_command))

);

assert(Pump\_State = pump\_command);

-- when to open valve - as long as liquid is above low sensor

eq valve\_command : bool = false ->

(

if (not SH\_Input and not SL\_Input) then false

else if (SH\_Input and SL\_Input) then true

else (valve\_command = pre(valve\_command))

);

assert(Valve\_State = valve\_command);

\*\*};

end Controller\_Tank1.Impl;

system Controller\_Tank2

features

SH\_Input: in data port Base\_Types::Boolean;

SM\_Input: in data port Base\_Types::Boolean;

SL\_Input: in data port Base\_Types::Boolean;

Valve\_Input\_State: in data port Base\_Types::Boolean;

Valve\_Output\_P\_State: out data port Base\_Types::Boolean;

Valve\_Output\_E\_State: out data port Base\_Types::Boolean;

annex agree {\*\*

-- assumption: goal is to keep the fluid between the sensors

-- Initial state: assume the valves are off (closed)

guarantee "The output production valve is initially closed" :

std.Init(not Valve\_Output\_P\_State);

guarantee "The output emergency valve is initially closed" :

std.Init(not Valve\_Output\_E\_State);

-- Liquid below Below Low sensor - just flow in

guarantee "When SL\_Input is False, the valves shall be Closed" :

true -> ((not SL\_Input) => ((not Valve\_Output\_P\_State) and (not Valve\_Output\_E\_State)));

-- Liquid below Below Mid sensor above low sensor

guarantee "When SM\_Input is False and the SL\_Input is true, the production valve shall be open and the emergency valve shall be closed" :

true -> ((not SM\_Input and SL\_Input) => (Valve\_Output\_P\_State and (not Valve\_Output\_E\_State)));

-- Liquid above high sensor

guarantee "When the SH\_Input is true, both valves will be open" :

true -> (SH\_Input => (Valve\_Output\_P\_State and Valve\_Output\_E\_State));

-- Liquid between mid and high sensors, previously emergency valve open

guarantee "When the SM\_Input is True and the SH\_Input is False and the emergency valve was previously open, the valves shall be Open" :

true -> ((not SH\_Input and SM\_Input and pre(Valve\_Output\_E\_State) and Valve\_Input\_State) => (Valve\_Output\_P\_State and Valve\_Output\_E\_State));

-- Liquid between mid and high sensors, previously emergency valve closed

guarantee "When the SM\_Input is True and the SH\_Input is False and the emergency valve was previously closed, the production valve shall be open and the emergency valve shall be closed" :

true -> ((not SH\_Input and SM\_Input and pre(not Valve\_Output\_E\_State)) => (Valve\_Output\_P\_State and not Valve\_Output\_E\_State));

-- System description

assume "If the high sensor is true, then the mid sensor is True" :

(SH\_Input => SM\_Input);

-- System description

assume "If the mid sensor is true, then the low sensor is True" :

(SM\_Input => SL\_Input);

\*\*};

end Controller\_Tank2;

system implementation Controller\_Tank2.Impl

annex agree {\*\*

eq production\_valve\_cmd : bool = false ->

(

if (SL\_Input) then true

else false

);

assert(Valve\_Output\_P\_State = production\_valve\_cmd);

eq emergency\_valve\_cmd : bool = false ->

(

if (SH\_Input or (SM\_Input and Valve\_Input\_State and pre(emergency\_valve\_cmd))) then true

else false

);

assert(Valve\_Output\_E\_State = emergency\_valve\_cmd);

\*\*};

end Controller\_Tank2.Impl;

system Controller\_Generic

features

SH\_Input: in data port Base\_Types::Boolean;

SL\_Input: in data port Base\_Types::Boolean;

Pump\_State: out data port Base\_Types::Boolean; -- commanded pump state and valve state

Valve\_State: out data port Base\_Types::Boolean;

annex agree {\*\*

-- assumption: goal is to keep the fluid between the sensors

-- Initial state: assume pump and valve are off

-- guarantee "Pump is initially Off and Valve is initially Closed" :

-- std.Initially(not Pump\_State and not Valve State);

guarantee "The pump is initially off":

std.Init(not Pump\_State);

guarantee "The valve is initially closed" :

std.Init(not Valve\_State);

-- Liquid below Below Low sensor - just flow in

guarantee "When SL\_Input is False, the Pump shall be on and Valve shall be Closed" :

true -> ((not SL\_Input) => ((not Valve\_State) and Pump\_State));

-- Liquid between sensors - decide flow to be in and out

guarantee "When the SL\_Input is True and the SH\_Input is False, the Pump shall be On and Valve shall be Open" :

true -> ((not SH\_Input and SL\_Input) => (Valve\_State and Pump\_State));

-- Liquid above high sensor - just flow out

guarantee "When the SH\_Input is true, the Pump shall be off and Valve shall be open" :

true -> (SH\_Input => (Valve\_State and (not Pump\_State)));

-- System description

assume "If the high sensor is true, then the low sensor is True" :

(SH\_Input => SL\_Input);

\*\*};

end Controller\_Generic;

system implementation Controller\_Generic.Impl

annex agree {\*\*

-- when to turn pump on - as long as liquid isn't at high sensor

eq pump\_command : bool = false -> (if (SH\_Input) then false else true);

assert(Pump\_State = pump\_command);

-- when to open valve - as long as liquid is above low sensor

eq valve\_command : bool = false -> (if (SL\_Input) then true else false);

assert(Valve\_State = valve\_command);

\*\*};

end Controller\_Generic.Impl;

end controller\_package;

==============================================================================

CtrlTanksExample.aadl

==============================================================================

package tank\_system

public

with Base\_Types;

with std;

with config;

with env\_package;

with controller\_package;

annex agree {\*\*

-- time rates

const init\_period\_s : int = config.init\_period\_s;

const high\_safety\_margin\_s : int = config.high\_safety\_margin\_s;

const low\_safety\_margin\_s : int = config.low\_safety\_margin\_s;

-- height parameters for Tank 1

const max\_tank1\_height\_m : real = config.max\_tank1\_height\_m;

const tank1\_empty\_m : real = config.tank1\_empty\_m;

-- safety parameters for Tank 1

const t1\_safety\_hi\_ht\_m : real = config.tank1\_safety\_hi\_ht\_m;

const t1\_safety\_lo\_ht\_m : real = config.tank1\_safety\_lo\_ht\_m;

-- height parameters for Tank 2

const max\_tank2\_height\_m : real = config.max\_tank2\_height\_m;

const tank2\_empty\_m : real = config.tank2\_empty\_m;

-- safety parameters for Tank 2

const t2\_safety\_hi\_ht\_m : real = config.tank2\_safety\_hi\_ht\_m;

const t2\_safety\_md\_ht\_m : real = config.tank2\_safety\_md\_ht\_m;

const t2\_safety\_lo\_ht\_m : real = config.tank2\_safety\_lo\_ht\_m;

\*\*};

system Two\_Tank\_Two\_Ctrl

features

Init\_Height\_Tank1: in data port Base\_Types::Float;

Init\_Height\_Tank2: in data port Base\_Types::Float;

Tank1\_Height: out data port Base\_Types::Float;

Tank2\_Height: out data port Base\_Types::Float;

annex agree {\*\*

eq t1\_sensor\_high : bool;

eq t1\_sensor\_low : bool;

-- Assumptions ----------------------------------

assume "Tank 1 shall Init be empty":

std.Init(Tank1\_Height = tank\_system.tank1\_empty\_m);

assume "Tank 2 shall Init be empty":

std.Init(Tank2\_Height = tank\_system.tank2\_empty\_m);

assume "No sensors fail":

true;

-------------------------------------------------

-- Safety Properties ----------------------------

guarantee "Tank 1 shall not overflow":

Tank1\_Height <= tank\_system.max\_tank1\_height\_m;

guarantee "Tank 2 shall not overflow":

Tank2\_Height <= tank\_system.max\_tank2\_height\_m;

-- guarantee "After an initial period n, the tank shall not be below the minimum threshold":

-- std.AfterN(Tank\_Height >= CtrlTanksExample.t1\_safety\_lo\_ht\_m, CtrlTanksExample.init\_period\_s);

guarantee "The height of the tank 1 liquid shall not be over the tank 1 high safety level for more than the specified length of time":

std.AfterNReset(Tank1\_Height < tank\_system.t1\_safety\_hi\_ht\_m,tank\_system.high\_safety\_margin\_s,(Tank1\_Height >= tank\_system.t1\_safety\_hi\_ht\_m));

guarantee "The height of the tank 1 liquid shall not be lower than the tank 1 low safety level (once it has already crossed it) for more than the specified length of time":

std.AfterNReset(Tank1\_Height > tank\_system.t1\_safety\_lo\_ht\_m,tank\_system.low\_safety\_margin\_s,std.AfterN(Tank1\_Height <= tank\_system.t1\_safety\_lo\_ht\_m,tank\_system.init\_period\_s));

guarantee "The height of the tank 2 liquid shall not be over the tank 2 high safety level for more than the specified length of time":

std.AfterNReset(Tank2\_Height < tank\_system.t2\_safety\_hi\_ht\_m,tank\_system.high\_safety\_margin\_s,(Tank2\_Height >= tank\_system.t2\_safety\_hi\_ht\_m));

guarantee "The height of the tank 2 liquid shall not be lower than the tank 2 low safety level (once it has already crossed it) for more than the specified length of time":

std.AfterNReset(Tank2\_Height > tank\_system.t2\_safety\_lo\_ht\_m,tank\_system.low\_safety\_margin\_s,std.AfterN(Tank2\_Height <= tank\_system.t2\_safety\_lo\_ht\_m,tank\_system.init\_period\_s));

-------------------------------------------------

-- Sanity Checks (Designed to Fail) -------------

-- guarantee "The tank height will not be bewteen the 2 safety values":

-- not((Tank\_Height > CtrlTanksExample.t1\_safety\_lo\_ht\_m) and (Tank\_Height < CtrlTanksExample.t1\_safety\_hi\_ht\_m));

--

-- guarantee "":

-- false;

-- guarantee "The emergency valve will never be on":

-- Tank2\_Height < config.tank2\_safety\_hi\_ht\_m;

-------------------------------------------------

\*\*};

end Two\_Tank\_Two\_Ctrl;

system implementation Two\_Tank\_Two\_Ctrl.Impl

subcomponents

C\_T1\_sub : system controller\_package::Controller\_Tank1.Impl;

C\_T2\_sub : system controller\_package::Controller\_Tank2.Impl;

E\_sub : system env\_package::Two\_Tank\_Environment.Impl;

connections

-- Tank Heights

E\_T1\_to\_OUT: port E\_sub.Tank1\_Height -> Tank1\_Height {Communication\_Properties::Timing => immediate;};

E\_T2\_to\_OUT: port E\_sub.Tank2\_Height -> Tank2\_Height {Communication\_Properties::Timing => immediate;};

-- Sensor interface

E\_sh\_T1\_to\_C\_T1\_sh: port E\_sub.Sensor\_T1\_High -> C\_T1\_sub.SH\_Input {Communication\_Properties::Timing => immediate;};

E\_sl\_T1\_to\_C\_T1\_sl: port E\_sub.Sensor\_T1\_Low -> C\_T1\_sub.SL\_Input {Communication\_Properties::Timing => immediate;};

E\_sh\_T2\_to\_C\_T2\_sh: port E\_sub.Sensor\_T2\_High -> C\_T2\_sub.SH\_Input {Communication\_Properties::Timing => immediate;};

E\_sm\_T2\_to\_C\_T2\_sm: port E\_sub.Sensor\_T2\_Mid -> C\_T2\_sub.SM\_Input {Communication\_Properties::Timing => immediate;};

E\_sl\_T2\_to\_C\_T2\_sl: port E\_sub.Sensor\_T2\_Low -> C\_T2\_sub.SL\_Input {Communication\_Properties::Timing => immediate;};

-- Control Outputs

C\_T1\_po\_to\_E\_po: port C\_T1\_sub.Pump\_State -> E\_sub.Pump\_On {Communication\_Properties::Timing => immediate;};

C\_T1\_vo\_to\_E\_T1\_vo: port C\_T1\_sub.Valve\_State -> E\_sub.Valve\_T1\_Open {Communication\_Properties::Timing => immediate;};

C\_T1\_vo\_to\_C\_T2\_v1: port C\_T1\_sub.Valve\_State -> C\_T2\_sub.Valve\_Input\_State {Communication\_Properties::Timing => immediate;};

C\_T2\_pvo\_to\_E\_T2\_pvo: port C\_T2\_sub.Valve\_Output\_P\_State -> E\_sub.Valve\_T2\_P\_Open {Communication\_Properties::Timing => immediate;};

C\_T2\_evo\_to\_E\_T2\_evo: port C\_T2\_sub.Valve\_Output\_E\_State -> E\_sub.Valve\_T2\_E\_Open {Communication\_Properties::Timing => immediate;};

annex agree {\*\*

assert (C\_T1\_sub.SH\_Input => C\_T1\_sub.SL\_Input);

assert (C\_T2\_sub.SH\_Input => C\_T2\_sub.SM\_Input);

assert (C\_T2\_sub.SM\_Input => C\_T2\_sub.SL\_Input);

assert (std.Init(not E\_sub.Pump\_On));

assert (std.Init(not E\_sub.Valve\_T1\_Open));

assert (std.Init(not E\_sub.Valve\_T2\_P\_Open));

assert (std.Init(not E\_sub.Valve\_T2\_E\_Open));

\*\*};

end Two\_Tank\_Two\_Ctrl.Impl;

system Single\_Tank

features

Init\_Height: in data port Base\_Types::Float;

Tank\_Height: out data port Base\_Types::Float;

annex agree {\*\*

-- Assumptions ----------------------------------

assume "Tank shall Init be empty":

std.Init(Tank\_Height = tank\_system.tank1\_empty\_m);

assume "No sensors fail":

true;

-------------------------------------------------

-- Safety Properties ----------------------------

guarantee "Tank shall not overflow":

Tank\_Height <= tank\_system.max\_tank1\_height\_m;

-- guarantee "After an initial period n, the tank shall not be below the minimum threshold":

-- std.AfterN(Tank\_Height >= CtrlTanksExample.t1\_safety\_lo\_ht\_m, CtrlTanksExample.init\_period\_s);

guarantee "The height of the liquid shall not be over the high safety level for more than the specified length of time":

std.AfterNReset(Tank\_Height < tank\_system.t1\_safety\_hi\_ht\_m,tank\_system.high\_safety\_margin\_s,(Tank\_Height >= tank\_system.t1\_safety\_hi\_ht\_m));

guarantee "The height of the liquid shall not be lower than the low safety level (once it has already crossed it) for more than the specified length of time":

std.AfterNReset(Tank\_Height > tank\_system.t1\_safety\_lo\_ht\_m,tank\_system.low\_safety\_margin\_s,std.AfterN(Tank\_Height <= tank\_system.t1\_safety\_lo\_ht\_m,tank\_system.init\_period\_s));

-------------------------------------------------

-- Sanity Checks (Designed to Fail) -------------

-- guarantee "The tank height will not be bewteen the 2 safety values":

-- not((Tank\_Height > CtrlTanksExample.t1\_safety\_lo\_ht\_m) and (Tank\_Height < CtrlTanksExample.t1\_safety\_hi\_ht\_m));

--

-- guarantee "":

-- false;

-------------------------------------------------

\*\*};

end Single\_Tank;

system implementation Single\_Tank.Impl

subcomponents

C\_sub : system controller\_package::Controller\_Tank1.Impl;

E\_sub : system env\_package::Single\_Tank\_Environment.Impl;

connections

E\_tv\_to\_OUT: port E\_sub.Tank\_Height -> Tank\_Height {Communication\_Properties::Timing => immediate;};

E\_sh\_to\_C\_sh: port E\_sub.Sensor\_High -> C\_sub.SH\_Input {Communication\_Properties::Timing => immediate;};

E\_sl\_to\_C\_sl: port E\_sub.Sensor\_Low -> C\_sub.SL\_Input {Communication\_Properties::Timing => immediate;};

C\_po\_to\_E\_po: port C\_sub.Pump\_State -> E\_sub.Pump\_On {Communication\_Properties::Timing => immediate;};

C\_vo\_to\_E\_vo: port C\_sub.Valve\_State -> E\_sub.Valve\_Open {Communication\_Properties::Timing => immediate;};

annex agree {\*\*

assert (C\_sub.SH\_Input => C\_sub.SL\_Input);

\*\*};

end Single\_Tank.Impl;

end tank\_system;

==============================================================================

Environment.aadl

==============================================================================

package env\_package

public

with Base\_Types;

with std;

with valve\_package;

with pump\_package;

with sensor\_package;

with tank\_package;

annex agree {\*\*

const tank1\_initial\_height\_m : real = tank\_package.tank1\_initial\_vol\_m3 / tank\_package.tank1\_cross\_section\_area\_m2;

const tank2\_initial\_height\_m : real = tank\_package.tank2\_initial\_vol\_m3 / tank\_package.tank2\_cross\_section\_area\_m2;

node Tank1\_Net\_Flow\_Constraint(Pump\_On: bool, Valve\_Open: bool, height: real) returns (in\_range: bool);

var

netFlow : real;

let

-- Need variable for initial tank height instead of stating 0.0

netFlow = std.Deltad(height, 0.0);

in\_range = std.After1(

-- if nothing is happening then the flow is 0.0 (no change)

if prev(not (Pump\_On or Valve\_Open),true) then (netFlow = 0.0)

else if prev((Pump\_On and not Valve\_Open),false) then

(netFlow >= 0.0 and netFlow <= config.t1\_pump\_max\_flow\_rate\_m3s)

else if prev((Pump\_On and Valve\_Open),false) then

if (pre(height = config.max\_tank1\_vol\_m3 / config.tank1\_cross\_section\_area\_m2) and (config.t1\_pump\_max\_flow\_rate\_m3s > config.t1\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (pre(height = 0.0) and (config.t1\_pump\_max\_flow\_rate\_m3s < config.t1\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (config.t1\_pump\_max\_flow\_rate\_m3s = config.t1\_valve\_max\_flow\_rate\_m3s) then

netFlow = 0.0

else

(netFlow <= config.t1\_pump\_max\_flow\_rate\_m3s) and netFlow >= -(config.t1\_valve\_max\_flow\_rate\_m3s)

else if prev(((not Pump\_On) and Valve\_Open),false) then

(netFlow >= -(config.t1\_valve\_max\_flow\_rate\_m3s) and netFlow <= 0.0)

else false);

tel;

node Tank2\_Net\_Flow\_Constraint(Valve\_In: bool, Valve\_P\_Out: bool, Valve\_E\_Out: bool, height: real) returns (in\_range: bool);

var

netFlow : real;

let

-- Need variable for initial tank height instead of stating 0.0

netFlow = std.Deltad(height, 0.0);

in\_range = std.After1(

-- if nothing is happening then the flow is 0.0 (no change)

if prev(not (Valve\_In or Valve\_P\_Out or Valve\_E\_Out),true) then (netFlow = 0.0)

else if prev((Valve\_In and not Valve\_P\_Out and not Valve\_E\_Out),false) then

(netFlow >= 0.0 and netFlow <= config.t1\_valve\_max\_flow\_rate\_m3s)

else if prev((Valve\_In and Valve\_P\_Out and not Valve\_E\_Out),false) then

if (pre(height = config.max\_tank2\_vol\_m3 / config.tank2\_cs\_area\_m2) and (config.t1\_valve\_max\_flow\_rate\_m3s > config.t2\_p\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (pre(height = 0.0) and (config.t1\_valve\_max\_flow\_rate\_m3s < config.t2\_p\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (config.t1\_valve\_max\_flow\_rate\_m3s = config.t2\_p\_valve\_max\_flow\_rate\_m3s) then

netFlow = 0.0

else

(netFlow <= config.t1\_valve\_max\_flow\_rate\_m3s) and netFlow >= -(config.t2\_p\_valve\_max\_flow\_rate\_m3s)

else if prev((Valve\_In and not Valve\_P\_Out and Valve\_E\_Out),false) then

if (pre(height = config.max\_tank2\_vol\_m3 / config.tank2\_cs\_area\_m2) and (config.t1\_valve\_max\_flow\_rate\_m3s > config.t2\_e\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (pre(height = 0.0) and (config.t1\_valve\_max\_flow\_rate\_m3s < config.t2\_e\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else if (config.t1\_valve\_max\_flow\_rate\_m3s = config.t2\_e\_valve\_max\_flow\_rate\_m3s) then

netFlow = 0.0

else

(netFlow <= config.t1\_valve\_max\_flow\_rate\_m3s) and netFlow >= -(config.t2\_e\_valve\_max\_flow\_rate\_m3s)

else if prev((Valve\_In and Valve\_P\_Out and Valve\_E\_Out),false) then

if (pre(height = config.max\_tank2\_vol\_m3 / config.tank2\_cs\_area\_m2) and (config.t1\_valve\_max\_flow\_rate\_m3s > (config.t2\_p\_valve\_max\_flow\_rate\_m3s + config.t2\_e\_valve\_max\_flow\_rate\_m3s))) then

netFlow = 0.0

else if (pre(height = 0.0) and (config.t1\_valve\_max\_flow\_rate\_m3s < (config.t2\_p\_valve\_max\_flow\_rate\_m3s + config.t2\_e\_valve\_max\_flow\_rate\_m3s))) then

netFlow = 0.0

else if (config.t1\_valve\_max\_flow\_rate\_m3s = (config.t2\_p\_valve\_max\_flow\_rate\_m3s + config.t2\_e\_valve\_max\_flow\_rate\_m3s)) then

netFlow = 0.0

else

(netFlow <= config.t1\_valve\_max\_flow\_rate\_m3s) and netFlow >= -(config.t2\_p\_valve\_max\_flow\_rate\_m3s + config.t2\_e\_valve\_max\_flow\_rate\_m3s)

else if prev(((not Valve\_In) and Valve\_P\_Out and Valve\_E\_Out),false) then

(netFlow >= -(config.t2\_p\_valve\_max\_flow\_rate\_m3s + config.t2\_e\_valve\_max\_flow\_rate\_m3s) and netFlow <= 0.0)

else if prev(((not Valve\_In) and Valve\_P\_Out and not Valve\_E\_Out),false) then

(netFlow >= -(config.t2\_p\_valve\_max\_flow\_rate\_m3s) and netFlow <= 0.0)

else if prev(((not Valve\_In) and not Valve\_P\_Out and Valve\_E\_Out),false) then

(netFlow >= -(config.t2\_e\_valve\_max\_flow\_rate\_m3s) and netFlow <= 0.0)

else false);

tel;

\*\*};

system Two\_Tank\_Environment

features

Pump\_On : in data port Base\_Types::Boolean;

Valve\_T1\_Open : in data port Base\_Types::Boolean;

Valve\_T2\_P\_Open : in data port Base\_Types::Boolean;

Valve\_T2\_E\_Open : in data port Base\_Types::Boolean;

Sensor\_T1\_High : out data port Base\_Types::Boolean;

Sensor\_T1\_Low : out data port Base\_Types::Boolean;

Sensor\_T2\_High : out data port Base\_Types::Boolean;

Sensor\_T2\_Mid : out data port Base\_Types::Boolean;

Sensor\_T2\_Low : out data port Base\_Types::Boolean;

Tank1\_Height : out data port Base\_Types::Float;

Tank2\_Height : out data port Base\_Types::Float;

Pump\_Flow : out data port Base\_Types::Float;

Valve\_T1\_Flow : out data port Base\_Types::Float;

-- Valve\_T2\_In\_Flow : out data port Base\_Types::Float;

Valve\_T2\_P\_Flow : out data port Base\_Types::Float;

Valve\_T2\_E\_Flow : out data port Base\_Types::Float;

annex agree {\*\*

eq tank1\_netFlow : real = Pump\_Flow - Valve\_T1\_Flow;

eq tank1\_netHtChg : real = tank1\_netFlow / config.tank1\_cross\_section\_area\_m2;

eq tank2\_netFlow : real = Valve\_T1\_Flow - Valve\_T2\_P\_Flow - Valve\_T2\_E\_Flow;

eq tank2\_netHtChg : real = tank2\_netFlow / config.tank2\_cs\_area\_m2;

-- Input Assumptions --------------------------------------

assume "The pump is Init off":

std.Init(not Pump\_On);

assume "The tank 1 valve is Init closed":

std.Init(not Valve\_T1\_Open);

assume "The tank 2 production valve is Init closed":

std.Init(not Valve\_T2\_P\_Open);

assume "The tank 2 emergency valve is Init closed":

std.Init(not Valve\_T2\_E\_Open);

-----------------------------------------------------------

-- Sensor Guarantees --------------------------------------

guarantee "The tank 1 sensor outputs will Init be false" :

std.Init(not Sensor\_T1\_High) and std.Init(not Sensor\_T1\_Low);

guarantee "The tank 2 sensor outputs will Init be false" :

std.Init(not Sensor\_T2\_High) and std.Init(not Sensor\_T2\_Mid) and std.Init(not Sensor\_T2\_Low);

guarantee "Tank 1: If the tank height is greater than or equal to the sensor high height then Sensor\_High is true" :

(false -> (Tank1\_Height >= sensor\_package.t1\_sensor\_hi\_height\_m)) => Sensor\_T1\_High;

guarantee "Tank 1: If the tank height is less than the sensor high height then Sensor\_High is false" :

(true -> (Tank1\_Height < sensor\_package.t1\_sensor\_hi\_height\_m)) => not Sensor\_T1\_High;

guarantee "Tank 1: If the tank height is greater than or equal to the sensor low height then Sensor\_Low is true" :

(false -> (Tank1\_Height >= sensor\_package.t1\_sensor\_lo\_height\_m)) => Sensor\_T1\_Low;

guarantee "Tank 1: If the tank height is less than the sensor low height then Sensor\_Low is false" :

(true -> (Tank1\_Height < sensor\_package.t1\_sensor\_lo\_height\_m)) => not Sensor\_T1\_Low;

guarantee "Tank 2: If the tank height is greater than or equal to the sensor high height then Sensor\_High is true" :

(false -> (Tank2\_Height >= sensor\_package.t2\_sensor\_hi\_height\_m)) => Sensor\_T2\_High;

guarantee "Tank 2: If the tank height is less than the sensor high height then Sensor\_High is false" :

(true -> (Tank2\_Height < sensor\_package.t2\_sensor\_hi\_height\_m)) => not Sensor\_T2\_High;

guarantee "Tank 2: If the tank height is greater than or equal to the sensor mid height then Sensor\_Mid is true" :

(false -> (Tank2\_Height >= sensor\_package.t2\_sensor\_md\_height\_m)) => Sensor\_T2\_Mid;

guarantee "Tank 2: If the tank height is less than the sensor mid height then Sensor\_Mid is false" :

(true -> (Tank2\_Height < sensor\_package.t2\_sensor\_md\_height\_m)) => not Sensor\_T2\_Mid;

guarantee "Tank 2: If the tank height is greater than or equal to the sensor low height then Sensor\_Low is true" :

(false -> (Tank2\_Height >= sensor\_package.t2\_sensor\_lo\_height\_m)) => Sensor\_T2\_Low;

guarantee "Tank 2: If the tank height is less than the sensor low height then Sensor\_Low is false" :

(true -> (Tank2\_Height < sensor\_package.t2\_sensor\_lo\_height\_m)) => not Sensor\_T2\_Low;

-----------------------------------------------------------

-- Tank 1 Guarantees --------------------------------------

guarantee "The tank 1 is Init empty":

std.Init(Tank1\_Height = 0.0);

guarantee "The tank 1 volume delta shall be bounded by the in\_range predicate" :

env\_package.Tank1\_Net\_Flow\_Constraint(Pump\_On, Valve\_T1\_Open, Tank1\_Height);

guarantee "The tank 1 current liquid height is equal to the previous tank 1 liquid height plus the additional tank 1 flow rate divided by the tank 1 cross-sectional area of the previous step" :

( (pre(Tank1\_Height) + prev(tank1\_netHtChg,0.0) >= 0.0) and (pre(Tank1\_Height) + prev(tank1\_netHtChg,0.0) <= (tank\_package.max\_tank1\_vol\_m3 / tank\_package.tank1\_cross\_section\_area\_m2)) )

=> (Tank1\_Height = (prev(Tank1\_Height,env\_package.tank1\_initial\_height\_m) + prev(tank1\_netHtChg,0.0)));

guarantee "If the pump is on and the valve is close the water level will rise (increase), Init true":

prev((Pump\_On and not Valve\_T1\_Open),false) => std.NonDecrease(Tank1\_Height, true);

guarantee "If the pump is off and the valve is open the water level will decrease or equal 0.0, Init true":

prev((not Pump\_On and Valve\_T1\_Open),false) => std.NonIncreasing(Tank1\_Height, true);

guarantee "If the pump and valve are both closed, the water level will remain the same" :

prev(not(Pump\_On or Valve\_T1\_Open),false) => std.NoChange(Tank1\_Height, true);

-----------------------------------------------------------

-- Tank 2 Guarantees --------------------------------------

guarantee "The tank 2 is Init empty":

std.Init(Tank2\_Height = 0.0);

guarantee "The tank 2 volume delta shall be bounded by the in\_range predicate" :

env\_package.Tank2\_Net\_Flow\_Constraint(Valve\_T1\_Open, Valve\_T2\_P\_Open, Valve\_T2\_E\_Open, Tank2\_Height);

guarantee "The tank 2 current liquid height is equal to the previous tank 2 liquid height plus the additional tank 2 flow rate divided by the tank 2 cross-sectional area of the previous step" :

( (pre(Tank2\_Height) + prev(tank2\_netHtChg,0.0) >= 0.0) and (pre(Tank2\_Height) + prev(tank2\_netHtChg,0.0) <= (tank\_package.max\_tank2\_vol\_m3 / tank\_package.tank2\_cross\_section\_area\_m2)) )

=> (Tank2\_Height = (prev(Tank2\_Height,env\_package.tank2\_initial\_height\_m) + prev(tank2\_netHtChg,0.0)));

guarantee "If the tank 1 valve is open and both tank 2 valves are closed the water level will rise (increase), Init true":

prev((Valve\_T1\_Open and not Valve\_T2\_P\_Open and not Valve\_T2\_E\_Open),false) => std.NonDecrease(Tank2\_Height, true);

guarantee "If the tank 1 valve and the tank 2 production valves are open and the tank 2 emergency valve is closed the water level will rise (increase), Init true":

prev((Valve\_T1\_Open and Valve\_T2\_P\_Open and not Valve\_T2\_E\_Open),false) => std.NonDecrease(Tank2\_Height, true);

guarantee "If the tank 1 valve and both tank 2 valves are open the water level will decrease or equal 0.0, Init true":

prev((Valve\_T1\_Open and Valve\_T2\_P\_Open and Valve\_T2\_E\_Open),false) => std.NonIncreasing(Tank2\_Height, true);

guarantee "If the tank 1 valve and tank 2 emergency valve are open and the tank 2 production valve is closed the water level will decrease or equal 0.0, Init true":

prev((Valve\_T1\_Open and not Valve\_T2\_P\_Open and Valve\_T2\_E\_Open),false) => std.NonIncreasing(Tank2\_Height, true);

guarantee "If the tank 1 valve is closed and either of tank 2 valves are open the water level will decrease or equal 0.0, Init true":

prev((not Valve\_T1\_Open and (Valve\_T2\_P\_Open or Valve\_T2\_E\_Open)),false) => std.NonIncreasing(Tank2\_Height, true);

guarantee "If all the valves are closed, the water level will remain the same" :

prev(not(Valve\_T1\_Open or Valve\_T2\_P\_Open or Valve\_T2\_E\_Open),false) => std.NoChange(Tank2\_Height, true);

-----------------------------------------------------------

-- Pump Guarantees ----------------------------------------

guarantee "If the pump is on then there should be a flow into the tank":

Pump\_On => Pump\_Flow = pump\_package.t1\_max\_flow\_rate\_m3s;

guarantee "If the pump is off then there should be no flow into the tank":

not Pump\_On => Pump\_Flow = 0.0;

-----------------------------------------------------------

-- Valve Guarantees ---------------------------------------

guarantee "If the tank 1 valve is open then there should be a flow from tank 1":

Valve\_T1\_Open => Valve\_T1\_Flow = valve\_package.t1\_max\_flow\_rate\_m3s;

guarantee "If the tank 1 valve is closed then there should be no flow from tank 1":

not Valve\_T1\_Open => Valve\_T1\_Flow = 0.0;

guarantee "If the tank 2 production valve is open then there should be a output flow equal to the max production flow":

Valve\_T2\_P\_Open => Valve\_T2\_P\_Flow = valve\_package.t2\_p\_max\_flow\_rate\_m3s;

guarantee "If the tank 2 production valve is closed then there should be no flow from the production valve":

not Valve\_T2\_P\_Open => Valve\_T2\_P\_Flow = 0.0;

guarantee "If the tank 2 emergency valve is open then there should be a output flow equal to the max emergency flow":

Valve\_T2\_E\_Open => Valve\_T2\_E\_Flow = valve\_package.t2\_e\_max\_flow\_rate\_m3s;

guarantee "If the tank 2 emergency valve is closed then there should be no flow from the emergency valve":

not Valve\_T2\_E\_Open => Valve\_T2\_E\_Flow = 0.0;

-----------------------------------------------------------

guarantee "If the sensor high is on then the sensor low should be on" :

Sensor\_T1\_High => Sensor\_T1\_Low;

-- Tank Connection Guarantees -----------------------------

-- guarantee "The output flow of tank 1 is the input flow of tank 2":

-- Valve\_T1\_Flow = Valve\_T2\_In\_Flow;

-----------------------------------------------------------

\*\*};

end Two\_Tank\_Environment;

system implementation Two\_Tank\_Environment.Impl

subcomponents

P\_sub : system pump\_package::T1\_Pump.Impl;

V\_T1\_sub : system valve\_package::T1\_Valve.Impl;

V\_T2\_P\_sub : system valve\_package::T2\_Production\_Valve.Impl;

V\_T2\_E\_sub : system valve\_package::T2\_Emergency\_Valve.Impl;

T1\_sub : system tank\_package::Tank\_1.UniformTankImpl;

T2\_sub : system tank\_package::Tank\_2.UniformTankImpl;

SH\_T1\_sub : system sensor\_package::T1\_Sensor\_High.H\_Impl;

SL\_T1\_sub : system sensor\_package::T1\_Sensor\_Low.L\_Impl;

SH\_T2\_sub : system sensor\_package::T2\_Sensor\_High.H\_Impl;

SM\_T2\_sub : system sensor\_package::T2\_Sensor\_Mid.M\_Impl;

SL\_T2\_sub : system sensor\_package::T2\_Sensor\_Low.L\_Impl;

connections

-- Tank Height Outputs

T1\_sub\_TO\_OUT : port T1\_sub.Liq\_Height\_m -> Tank1\_Height {Communication\_Properties::Timing => immediate;};

T2\_sub\_TO\_OUT : port T2\_sub.Liq\_Height\_m -> Tank2\_Height {Communication\_Properties::Timing => immediate;};

-- Sensor Input Data

T1\_sub\_TO\_SH\_T1\_sub : port T1\_sub.Liq\_Height\_m -> SH\_T1\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

T1\_sub\_TO\_SL\_T1\_sub : port T1\_sub.Liq\_Height\_m -> SL\_T1\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

T2\_sub\_TO\_SH\_T2\_sub : port T2\_sub.Liq\_Height\_m -> SH\_T2\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

T2\_sub\_TO\_SM\_T2\_sub : port T2\_sub.Liq\_Height\_m -> SM\_T2\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

T2\_sub\_TO\_SL\_T2\_sub : port T2\_sub.Liq\_Height\_m -> SL\_T2\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

-- Sensor Outputs

SH\_T1\_sub\_TO\_OUT : port SH\_T1\_sub.Sensor\_Value -> Sensor\_T1\_High {Communication\_Properties::Timing => immediate;};

SL\_T1\_sub\_TO\_OUT : port SL\_T1\_sub.Sensor\_Value -> Sensor\_T1\_Low {Communication\_Properties::Timing => immediate;};

SH\_T2\_sub\_TO\_OUT : port SH\_T2\_sub.Sensor\_Value -> Sensor\_T2\_High {Communication\_Properties::Timing => immediate;};

SM\_T2\_sub\_TO\_OUT : port SM\_T2\_sub.Sensor\_Value -> Sensor\_T2\_Mid {Communication\_Properties::Timing => immediate;};

SL\_T2\_sub\_TO\_OUT : port SL\_T2\_sub.Sensor\_Value -> Sensor\_T2\_Low {Communication\_Properties::Timing => immediate;};

-- Pump Control

IN\_po\_TO\_P\_sub : port Pump\_On -> P\_sub.Pump\_State {Communication\_Properties::Timing => immediate;};

-- Valve Control

IN\_vo\_TO\_V\_T1\_sub : port Valve\_T1\_Open -> V\_T1\_sub.Valve\_State {Communication\_Properties::Timing => immediate;};

IN\_vo\_TO\_V\_T2\_P\_sub : port Valve\_T2\_P\_Open -> V\_T2\_P\_sub.Valve\_State {Communication\_Properties::Timing => immediate;};

IN\_vo\_TO\_V\_T2\_E\_sub : port Valve\_T2\_E\_Open -> V\_T2\_E\_sub.Valve\_State {Communication\_Properties::Timing => immediate;};

-- Tank 1 Flows

P\_sub\_TO\_T1\_sub : port P\_sub.Flow\_Out -> T1\_sub.Flow\_In {Communication\_Properties::Timing => immediate;};

V\_T1\_sub\_TO\_T1\_sub : port V\_T1\_sub.Flow\_Out -> T1\_sub.Flow\_Out {Communication\_Properties::Timing => immediate;};

-- Tank 2 Flows

V\_T1\_sub\_TO\_T2\_sub : port V\_T1\_sub.Flow\_Out -> T2\_sub.Flow\_In {Communication\_Properties::Timing => immediate;};

V\_T2\_P\_sub\_TO\_T2\_sub : port V\_T2\_P\_sub.Flow\_Out -> T2\_sub.Flow\_Production\_Out {Communication\_Properties::Timing => immediate;};

V\_T2\_E\_sub\_TO\_T2\_sub : port V\_T2\_E\_sub.Flow\_Out -> T2\_sub.Flow\_Emergency\_Out {Communication\_Properties::Timing => immediate;};

-- Data to the environment

P\_sub\_TO\_OUT : port P\_sub.Flow\_Out -> Pump\_Flow {Communication\_Properties::Timing => immediate;};

V\_T1\_sub\_TO\_OUT : port V\_T1\_sub.Flow\_Out -> Valve\_T1\_Flow {Communication\_Properties::Timing => immediate;};

V\_T2\_P\_sub\_TO\_OUT : port V\_T2\_P\_sub.Flow\_Out -> Valve\_T2\_P\_Flow {Communication\_Properties::Timing => immediate;};

V\_T2\_E\_sub\_TO\_OUT : port V\_T2\_E\_sub.Flow\_Out -> Valve\_T2\_E\_Flow {Communication\_Properties::Timing => immediate;};

annex agree {\*\*

eq dummy : bool = true;

\*\*};

end Two\_Tank\_Environment.Impl;

system Single\_Tank\_Environment

features

Pump\_On : in data port Base\_Types::Boolean;

Valve\_Open : in data port Base\_Types::Boolean;

Sensor\_High : out data port Base\_Types::Boolean;

Sensor\_Low : out data port Base\_Types::Boolean;

Tank\_Height : out data port Base\_Types::Float;

Pump\_Flow : out data port Base\_Types::Float;

Valve\_Flow : out data port Base\_Types::Float;

annex agree {\*\*

-- Sensor Guarantees --------------------------------------

guarantee "The sensor outputs will Init be false" :

std.Init(not Sensor\_High) and std.Init(not Sensor\_Low);

guarantee "If the tank height is greater than or equal to the sensor high height then Sensor\_High is true" :

(false -> (Tank\_Height >= sensor\_package.t1\_sensor\_hi\_height\_m)) => Sensor\_High;

guarantee "If the tank height is less than the sensor high height then Sensor\_High is false" :

(true -> (Tank\_Height < sensor\_package.t1\_sensor\_hi\_height\_m)) => not Sensor\_High;

guarantee "If the tank height is greater than or equal to the sensor low height then Sensor\_Low is true" :

(false -> (Tank\_Height >= sensor\_package.t1\_sensor\_lo\_height\_m)) => Sensor\_Low;

guarantee "If the tank height is less than the sensor low height then Sensor\_Low is false" :

(true -> (Tank\_Height < sensor\_package.t1\_sensor\_lo\_height\_m)) => not Sensor\_Low;

-----------------------------------------------------------

-- Tank Guarantees ----------------------------------------

guarantee "The tank is Init empty":

std.Init(Tank\_Height = 0.0);

guarantee "The tank volume delta shall be bounded by the in\_range predicate" :

env\_package.Tank1\_Net\_Flow\_Constraint(Pump\_On, Valve\_Open, Tank\_Height);

guarantee "The current liquid height is equal to the previous liquid height plus the additional flow rate / cross-sectional area of the previous step" :

( (pre(Tank\_Height) + prev(((Pump\_Flow - Valve\_Flow)/tank\_package.tank1\_cross\_section\_area\_m2),0.0) >= 0.0) and (pre(Tank\_Height) + prev(((Pump\_Flow - Valve\_Flow)/tank\_package.tank1\_cross\_section\_area\_m2),0.0) <= (tank\_package.max\_tank1\_vol\_m3 / tank\_package.tank1\_cross\_section\_area\_m2)) )

=> (Tank\_Height = (prev(Tank\_Height,env\_package.tank1\_initial\_height\_m) + prev(((Pump\_Flow - Valve\_Flow)/tank\_package.tank1\_cross\_section\_area\_m2),0.0)));

guarantee "If the pump is on and the valve is close the water level will rise (increase), Init increasing":

prev((Pump\_On and not Valve\_Open),false) => std.NonDecrease(Tank\_Height, true);

guarantee "If the pump is off and the valve is open the water level will decrease or equal 0.0, Init true":

prev((not Pump\_On and Valve\_Open),false) => std.NonIncreasing(Tank\_Height, true);

guarantee "If the pump and valve are both closed, the water level will remain the same" :

prev(not(Pump\_On or Valve\_Open),false) => std.NoChange(Tank\_Height, true);

-----------------------------------------------------------

-- Pump Guarantees ----------------------------------------

guarantee "If the pump is on then there should be a flow into the tank":

Pump\_On => Pump\_Flow = pump\_package.t1\_max\_flow\_rate\_m3s;

guarantee "If the pump is off then there should be no flow into the tank":

not Pump\_On => Pump\_Flow = 0.0;

-----------------------------------------------------------

-- Valve Guarantees ---------------------------------------

guarantee "If the valve is open then there should be a flow from the tank":

Valve\_Open => Valve\_Flow = valve\_package.t1\_max\_flow\_rate\_m3s;

guarantee "If the valve is closed then there should be no flow from the tank":

not Valve\_Open => Valve\_Flow = 0.0;

-----------------------------------------------------------

\*\*};

end Single\_Tank\_Environment;

system implementation Single\_Tank\_Environment.Impl

subcomponents

P\_sub : system pump\_package::T1\_Pump.Impl;

V\_sub : system valve\_package::T1\_Valve.Impl;

T\_sub : system tank\_package::Tank\_1.UniformTankImpl;

SH\_sub : system sensor\_package::T1\_Sensor\_High.H\_Impl;

SL\_sub : system sensor\_package::T1\_Sensor\_Low.L\_Impl;

connections

T\_sub\_TO\_OUT : port T\_sub.Liq\_Height\_m -> Tank\_Height {Communication\_Properties::Timing => immediate;};

T\_sub\_TO\_SH\_sub : port T\_sub.Liq\_Height\_m -> SH\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

T\_sub\_TO\_SL\_sub : port T\_sub.Liq\_Height\_m -> SL\_sub.Tank\_Height {Communication\_Properties::Timing => immediate;};

SH\_sub\_TO\_OUT : port SH\_sub.Sensor\_Value -> Sensor\_High {Communication\_Properties::Timing => immediate;};

SL\_sub\_TO\_OUT : port SL\_sub.Sensor\_Value -> Sensor\_Low {Communication\_Properties::Timing => immediate;};

IN\_po\_TO\_P\_sub : port Pump\_On -> P\_sub.Pump\_State {Communication\_Properties::Timing => immediate;};

IN\_vo\_TO\_V\_sub : port Valve\_Open -> V\_sub.Valve\_State {Communication\_Properties::Timing => immediate;};

P\_sub\_TO\_T\_sub : port P\_sub.Flow\_Out -> T\_sub.Flow\_In {Communication\_Properties::Timing => immediate;};

V\_sub\_TO\_T\_sub : port V\_sub.Flow\_Out -> T\_sub.Flow\_Out {Communication\_Properties::Timing => immediate;};

P\_sub\_TO\_OUT : port P\_sub.Flow\_Out -> Pump\_Flow {Communication\_Properties::Timing => immediate;};

V\_sub\_TO\_OUT : port V\_sub.Flow\_Out -> Valve\_Flow {Communication\_Properties::Timing => immediate;};

annex agree {\*\*

eq dummy : bool = true;

\*\*};

end Single\_Tank\_Environment.Impl;

end env\_package;

==============================================================================

pump.aadl

==============================================================================

package pump\_package

public

with Base\_Types;

with config;

annex agree {\*\*

const t1\_max\_flow\_rate\_m3s : real = config.t1\_pump\_max\_flow\_rate\_m3s;

const t1\_zero\_flow\_rate\_m3s : real = config.t1\_pump\_zero\_flow\_rate\_m3s;

\*\*};

system T1\_Pump

features

Pump\_State: in data port Base\_Types::Boolean;

Flow\_Out: out data port Base\_Types::Float;

annex agree {\*\*

guarantee "If the pump is on then the flow through the pump is the max flow rate" :

Pump\_State => (Flow\_Out = pump\_package.t1\_max\_flow\_rate\_m3s);

guarantee "If the pump is off then the flow rate through the pump is 0.0" :

(not Pump\_State) => (Flow\_Out = pump\_package.t1\_zero\_flow\_rate\_m3s);

guarantee "If the Flow in is the max pump rate, then the pump is receiving an ON signal" :

(Flow\_Out = pump\_package.t1\_max\_flow\_rate\_m3s) => Pump\_State;

guarantee "If the Flow in is 0.0, then the pump is receiving on OFF signal" :

(Flow\_Out = pump\_package.t1\_zero\_flow\_rate\_m3s) => not Pump\_State;

\*\*};

end T1\_Pump;

system implementation T1\_Pump.Impl

annex agree {\*\*

eq flow\_rate : real = if (Pump\_State) then pump\_package.t1\_max\_flow\_rate\_m3s else pump\_package.t1\_zero\_flow\_rate\_m3s;

-- This is to assert that the output acts as it should

assert(Flow\_Out = flow\_rate);

\*\*};

end T1\_Pump.Impl;

end pump\_package;

==============================================================================

sensor.aadl

==============================================================================

package sensor\_package

public

with Base\_Types;

with std;

with config;

annex agree {\*\*

-- Tank 1 specifications ----------------------------

const t1\_sensor\_lo\_height\_m : real = config.t1\_sensor\_lo\_height\_m;

const t1\_sensor\_hi\_height\_m : real = config.t1\_sensor\_hi\_height\_m;

-----------------------------------------------------

-- Tank 2 specifications ----------------------------

const t2\_sensor\_hi\_height\_m : real = config.t2\_sensor\_hi\_height\_m;

const t2\_sensor\_md\_height\_m : real = config.t2\_sensor\_md\_height\_m;

const t2\_sensor\_lo\_height\_m : real = config.t2\_sensor\_lo\_height\_m;

-----------------------------------------------------

\*\*};

-- Tank 2 -------------------------------------------

system T2\_Sensor\_High

features

Tank\_Height : in data port Base\_Types::Float;

Sensor\_Value: out data port Base\_Types::Boolean;

Sensor\_Height : out data port Base\_Types::Float;

annex agree {\*\*

eq sensor\_height : real;

guarantee "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true" :

(false -> (Tank\_Height >= sensor\_height)) => Sensor\_Value;

guarantee "If the tank height is less than the sensor height then Sensor\_Value is false" :

(true -> (Tank\_Height < sensor\_height)) => not Sensor\_Value;

guarantee "The sensor should be false Init":

std.Init(Sensor\_Value = false);

guarantee "The Sensor\_Height is the same as the specified sensor height":

Sensor\_Height = sensor\_package.t2\_sensor\_hi\_height\_m;

guarantee "The sensor\_height is the same as the specified sensor height":

sensor\_height = sensor\_package.t2\_sensor\_hi\_height\_m;

\*\*};

end T2\_Sensor\_High;

system implementation T2\_Sensor\_High.H\_Impl

annex agree {\*\*

eq sensor\_hi\_on : bool = false -> (Tank\_Height >= sensor\_package.t2\_sensor\_hi\_height\_m);

assert(sensor\_height = sensor\_package.t2\_sensor\_hi\_height\_m);

assert(Sensor\_Value = sensor\_hi\_on);

assert(Sensor\_Height = sensor\_height);

\*\*};

end T2\_Sensor\_High.H\_Impl;

system T2\_Sensor\_Mid

features

Tank\_Height : in data port Base\_Types::Float;

Sensor\_Value: out data port Base\_Types::Boolean;

Sensor\_Height : out data port Base\_Types::Float;

annex agree {\*\*

eq sensor\_height : real;

guarantee "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true" :

(false -> (Tank\_Height >= sensor\_height)) => Sensor\_Value;

guarantee "If the tank height is less than the sensor height then Sensor\_Value is false" :

(true -> (Tank\_Height < sensor\_height)) => not Sensor\_Value;

guarantee "The sensor should be false Init":

std.Init(Sensor\_Value = false);

guarantee "The Sensor\_Height is the same as the specified sensor height":

Sensor\_Height = sensor\_package.t2\_sensor\_md\_height\_m;

guarantee "The sensor\_height is the same as the specified sensor height":

sensor\_height = sensor\_package.t2\_sensor\_md\_height\_m;

\*\*};

end T2\_Sensor\_Mid;

system implementation T2\_Sensor\_Mid.M\_Impl

annex agree {\*\*

eq sensor\_md\_on : bool = false -> (Tank\_Height >= sensor\_package.t2\_sensor\_md\_height\_m);

assert(sensor\_height = sensor\_package.t2\_sensor\_md\_height\_m);

assert(Sensor\_Value = sensor\_md\_on);

assert(Sensor\_Height = sensor\_height);

\*\*};

end T2\_Sensor\_Mid.M\_Impl;

system T2\_Sensor\_Low

features

Tank\_Height : in data port Base\_Types::Float;

Sensor\_Value: out data port Base\_Types::Boolean;

Sensor\_Height : out data port Base\_Types::Float;

annex agree {\*\*

eq sensor\_height : real;

guarantee "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true" :

(false -> (Tank\_Height >= sensor\_height)) => Sensor\_Value;

guarantee "If the tank height is less than the sensor height then Sensor\_Value is false" :

(true -> (Tank\_Height < sensor\_height)) => not Sensor\_Value;

guarantee "The sensor should be false Init":

std.Init(Sensor\_Value = false);

guarantee "The Sensor\_Height is the same as the specified sensor height":

Sensor\_Height = sensor\_package.t2\_sensor\_lo\_height\_m;

guarantee "The sensor\_height is the same as the specified sensor height":

sensor\_height = sensor\_package.t2\_sensor\_lo\_height\_m;

\*\*};

end T2\_Sensor\_Low;

system implementation T2\_Sensor\_Low.L\_Impl

annex agree {\*\*

eq sensor\_low\_on : bool = false -> (Tank\_Height >= sensor\_package.t2\_sensor\_lo\_height\_m);

assert(sensor\_height = sensor\_package.t2\_sensor\_lo\_height\_m);

assert(Sensor\_Value = sensor\_low\_on);

assert(Sensor\_Height = sensor\_height);

\*\*};

end T2\_Sensor\_Low.L\_Impl;

-----------------------------------------------------

-- Tank 1 -------------------------------------------

system T1\_Sensor\_High

features

Tank\_Height : in data port Base\_Types::Float;

Sensor\_Value: out data port Base\_Types::Boolean;

Sensor\_Height : out data port Base\_Types::Float;

annex agree {\*\*

eq sensor\_height : real;

guarantee "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true" :

(false -> (Tank\_Height >= sensor\_height)) => Sensor\_Value;

guarantee "If the tank height is less than the sensor height then Sensor\_Value is false" :

(true -> (Tank\_Height < sensor\_height)) => not Sensor\_Value;

guarantee "The sensor should be false Init":

std.Init(Sensor\_Value = false);

guarantee "The Sensor\_Height is the same as the specified sensor height":

Sensor\_Height = sensor\_package.t1\_sensor\_hi\_height\_m;

guarantee "The sensor\_height is the same as the specified sensor height":

sensor\_height = sensor\_package.t1\_sensor\_hi\_height\_m;

\*\*};

end T1\_Sensor\_High;

system implementation T1\_Sensor\_High.H\_Impl

annex agree {\*\*

eq sensor\_hi\_on : bool = false -> (Tank\_Height >= sensor\_package.t1\_sensor\_hi\_height\_m);

assert(sensor\_height = sensor\_package.t1\_sensor\_hi\_height\_m);

assert(Sensor\_Value = sensor\_hi\_on);

assert(Sensor\_Height = sensor\_height);

\*\*};

end T1\_Sensor\_High.H\_Impl;

system T1\_Sensor\_Low

features

Tank\_Height : in data port Base\_Types::Float;

Sensor\_Value: out data port Base\_Types::Boolean;

Sensor\_Height : out data port Base\_Types::Float;

annex agree {\*\*

eq sensor\_height : real;

guarantee "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true" :

(false -> (Tank\_Height >= sensor\_height)) => Sensor\_Value;

guarantee "If the tank height is less than the sensor height then Sensor\_Value is false" :

(true -> (Tank\_Height < sensor\_height)) => not Sensor\_Value;

guarantee "The sensor should be false Init":

std.Init(Sensor\_Value = false);

guarantee "The Sensor\_Height is the same as the specified sensor height":

Sensor\_Height = sensor\_package.t1\_sensor\_lo\_height\_m;

guarantee "The sensor\_height is the same as the specified sensor height":

sensor\_height = sensor\_package.t1\_sensor\_lo\_height\_m;

\*\*};

end T1\_Sensor\_Low;

system implementation T1\_Sensor\_Low.L\_Impl

annex agree {\*\*

eq sensor\_low\_on : bool = false -> (Tank\_Height >= sensor\_package.t1\_sensor\_lo\_height\_m);

assert(sensor\_height = sensor\_package.t1\_sensor\_lo\_height\_m);

assert(Sensor\_Value = sensor\_low\_on);

assert(Sensor\_Height = sensor\_height);

\*\*};

end T1\_Sensor\_Low.L\_Impl;

-----------------------------------------------------

end sensor\_package;

==============================================================================

std.aadl

==============================================================================

package std

public

annex agree {\*\*

node RisingEdge(i: bool, init: bool) returns (o: bool);

let

o = init -> i and (not pre(i));

tel;

node re(i: bool, init: bool) returns (o: bool);

let

o = RisingEdge(i, init);

tel;

node FallingEdge(i: bool, init: bool) returns (o: bool);

let

o = init -> (not i) and pre(i);

tel;

node fe(i: bool, init: bool) returns (o: bool);

let

o = FallingEdge(i, init);

tel;

node Init(i: bool) returns (o: bool);

let

o = i -> true;

tel;

node After1(i: bool) returns (o: bool);

let

o = true -> i;

tel;

node Increase(i: real, init: bool) returns (o: bool);

let

o = init -> (i > pre(i));

tel;

node NonDecrease(i: real, init: bool) returns (o: bool);

let

o = init -> (i >= pre(i));

tel;

node Decrease(i: real, init: bool) returns (o: bool);

let

o = init -> (i < pre(i));

tel;

node NonIncreasing(i: real, init: bool) returns (o: bool);

let

o = init -> (i <= pre(i));

tel;

node NoChange(i: real, init: bool) returns (o: bool);

let

o = init -> (i = pre(i));

tel;

node AbsoluteValue(i: real) returns (o: real);

let

o = if (i >= 0.0) then i else -i;

tel;

node abs(i: real) returns (o: real);

let

o = AbsoluteValue(i);

tel;

node Deltad(i: real, init: real) returns (o: real);

let

o = init -> (i - pre(i));

tel;

node timer(set: bool, reset: bool) returns (time\_val: int);

let

time\_val =

if reset then 0 else

prev(time\_val, 0) + (if set then 1 else 0);

tel;

node AfterN(i: bool, n: int) returns (o: bool);

let

o = if (timer(true, false) >= n) then i else true;

tel;

node AfterNReset(i: bool, n: int, reset: bool) returns (o: bool);

let

o = if (timer(true, false->reset) >= n) then i else true;

tel;

node WithinN(wnEvent: bool, i: bool, n: int) returns (o: bool);

var

after\_event: bool;

time: int;

let

after\_event = if i then false else if wnEvent then true else

(false -> pre(after\_event));

time = if i then 0 else

(0 -> pre(time)) + if after\_event then 1 else 0;

o = time <= n;

tel;

node While(i: bool) returns (o: bool);

let

o = i -> (i and pre(i)) ;

tel;

node Historically(i: bool) returns (o: bool);

let

o = i -> i and (pre(o));

tel;

node Occurred(i: bool) returns (o: bool);

let

o = i -> i or (pre(o));

tel;

-- Q: do we want a property to be guaranteed false after it is

-- initially falsified?

node WeakUntil(i: bool, j: bool) returns (o: bool);

let

o = Historically(i) or Occurred(j);

tel;

-- node Precedes(i: bool, j: bool) returns (o: bool);

-- let

-- o = Occurred(j) => (false -> pre(Occurred(i)));

-- tel;

node true\_interval(signal: bool) returns (time\_val: int);

let

time\_val = timer(signal, not signal);

tel;

-- One of the property patterns from Lucas.

node after\_p\_a\_precedes\_b(p : bool, a : bool, b : bool) returns (holds : bool);

var

state : int;

pre\_state : int;

prop1 : bool;

let

pre\_state = 0 -> pre(state);

state =

if (pre\_state = 0)

then

if (not p)

then 0

else if (p and a)

then 0

else if (p and not a and not b)

then 1

else if (p and not a and b)

then 2

else -1

else if (pre\_state = 1)

then

if a

then 0

else if (not a and not b)

then 1

else if (not a and b)

then 2

else -1

else if (pre\_state = 2)

then 2

else -1;

holds = (state <> 2);

prop1 = (state <> -1);

lemma "range" : state >= 0 and state <= 2;

--%PROPERTY prop1;

tel;

node before\_p\_always\_a(p : bool, a : bool) returns (holds : bool);

var

state : int;

pre\_state : int;

prop1 : bool;

let

pre\_state = 0 -> pre(state);

state =

if (pre\_state = 0)

then

if p

then 1

else if(not p and a)

then 0

else if(not p and not a)

then 2

else -1

else if(pre\_state = 1)

then 1

else if(pre\_state = 2)

then

if(not p)

then 2

else if p

then 3

else -1

else if(pre\_state = 3)

then 3

else -1;

lemma "range": state >= 0 and state <= 3;

holds = (state <> 3);

prop1 = (state <> -1);

tel;

node IncreaseInt(i: int) returns (o: bool);

let

o = true -> (i > pre(i));

tel;

node DecreaseInt(i: int) returns (o: bool);

let

o = true -> (i < pre(i));

tel;

node AbsoluteValueInt(i: int) returns (o: int);

let

o = if (i >= 0) then i else -i;

tel;

node absi(i: int) returns (o: int);

let

o = AbsoluteValueInt(i);

tel;

\*\*} ;

end std;

==============================================================================

tank.aadl

==============================================================================

package tank\_package

public

with Base\_Types;

with config;

annex agree {\*\*

-- Tank 1 specifications ----------------------------

const tank1\_initial\_vol\_m3 : real = config.tank1\_initial\_vol\_m3;

const max\_tank1\_vol\_m3 : real = config.max\_tank1\_vol\_m3;

const tank1\_cross\_section\_area\_m2 : real = config.tank1\_cross\_section\_area\_m2;

-----------------------------------------------------

-- Tank 2 specifications ----------------------------

const tank2\_initial\_vol\_m3 : real = config.tank2\_initial\_vol\_m3;

const max\_tank2\_vol\_m3 : real = config.max\_tank2\_vol\_m3;

const tank2\_cross\_section\_area\_m2 : real = config.tank2\_cs\_area\_m2;

-----------------------------------------------------

\*\*};

system Tank\_1

features

Flow\_In: in data port Base\_Types::Float;

Flow\_Out: in data port Base\_Types::Float;

Liq\_Height\_m: out data port Base\_Types::Float;

annex agree {\*\*

eq current\_vol : real;

eq max\_tank\_height : real;

eq current\_height : real;

guarantee "The initial volume of tank 1 should be equivalent to the specified initial tank 1 volume" :

std.Init(current\_vol = tank\_package.tank1\_initial\_vol\_m3);

guarantee "If the current tank 1 volume plus the tank 1 net flow is greater than the tank 1 maximum volume then tank 1 is full":

(prev(current\_vol, tank\_package.tank1\_initial\_vol\_m3) + prev(Flow\_In - Flow\_Out,0.0) > tank\_package.max\_tank1\_vol\_m3) => (current\_vol = tank\_package.max\_tank1\_vol\_m3);

guarantee "If the current tank 1 volume plus the tank 1 net flow is less than the tank 1 minimum volume then tank 1 is empty" :

(prev(current\_vol, tank\_package.tank1\_initial\_vol\_m3) + prev(Flow\_In - Flow\_Out,0.0) < 0.0) => (current\_vol = 0.0);

guarantee "If the current tank 1 volume plus the tank 1 net flow is greater than empty and less than the tank 1 maximum volume then the tank 1 volume changes by the tank 1 net flow":

(prev(current\_vol, tank\_package.tank1\_initial\_vol\_m3) + prev(Flow\_In - Flow\_Out,0.0) <= tank\_package.max\_tank1\_vol\_m3) and (prev(current\_vol, tank\_package.tank1\_initial\_vol\_m3) + prev(Flow\_In - Flow\_Out,0.0) >= 0.0)

=> (current\_vol = prev(current\_vol, tank\_package.tank1\_initial\_vol\_m3) + prev(Flow\_In - Flow\_Out,0.0));

guarantee "Current height of tank 1 is equal or less than the max tank 1 height" :

Liq\_Height\_m <= max\_tank\_height;

guarantee "Current height of tank 1 is non negative" :

Liq\_Height\_m >= 0.0;

guarantee "The tank 1 current height is equal to the tank 1 current volume divided by the tank 1 cross sectional area" :

Liq\_Height\_m = current\_vol / tank\_package.tank1\_cross\_section\_area\_m2;

guarantee "The max tank 1 height is equal to the max tank 1 volume divided by tank 1 cross sectional area":

max\_tank\_height = tank\_package.max\_tank1\_vol\_m3 / tank\_package.tank1\_cross\_section\_area\_m2;

guarantee "The current tank 1 height is the current tank 1 volume divided by the tank 1 cross sectional area":

current\_height = (current\_vol / tank\_package.tank1\_cross\_section\_area\_m2);

guarantee "The output height of tank 1 should equal the calculated height of tank 1":

Liq\_Height\_m = current\_height;

\*\*};

end Tank\_1;

system implementation Tank\_1.UniformTankImpl

annex agree{\*\*

eq delta\_vol : real = prev(Flow\_In - Flow\_Out,0.0);

eq tank\_vol\_calc : real = ((tank\_package.tank1\_initial\_vol\_m3 -> pre(current\_vol)) + delta\_vol);

eq tank\_vol\_overflow\_underflow : real = if (tank\_vol\_calc > tank\_package.max\_tank1\_vol\_m3) then tank\_package.max\_tank1\_vol\_m3 else if (tank\_vol\_calc < 0.0) then 0.0 else tank\_vol\_calc;

assert(current\_vol = tank\_vol\_overflow\_underflow);

assert(max\_tank\_height = (tank\_package.max\_tank1\_vol\_m3 / tank\_package.tank1\_cross\_section\_area\_m2));

assert(current\_height = (current\_vol / tank\_package.tank1\_cross\_section\_area\_m2));

-- Output

assert(Liq\_Height\_m = current\_height);

\*\*};

end Tank\_1.UniformTankImpl;

system Tank\_2

features

Flow\_In: in data port Base\_Types::Float;

Flow\_Production\_Out: in data port Base\_Types::Float;

Flow\_Emergency\_Out: in data port Base\_Types::Float;

Liq\_Height\_m: out data port Base\_Types::Float;

annex agree {\*\*

eq current\_vol : real;

eq max\_tank\_height : real;

eq current\_height : real;

eq netFlow : real = (Flow\_In - Flow\_Production\_Out - Flow\_Emergency\_Out);

guarantee "The initial volume of tank 2 should be equivalent to the specified initial tank 2 volume" :

std.Init(current\_vol = tank\_package.tank2\_initial\_vol\_m3);

guarantee "If the current tank 2 volume plus the tank 2 net flow is greater than the tank 2 maximum volume then tank 2 is full":

(prev(current\_vol, tank\_package.tank2\_initial\_vol\_m3) + prev(netFlow,0.0) > tank\_package.max\_tank2\_vol\_m3) => (current\_vol = tank\_package.max\_tank2\_vol\_m3);

guarantee "If the current tank 2 volume plus the tank 2 net flow is less than the tank 2 minimum volume then tank 2 is empty" :

(prev(current\_vol, tank\_package.tank2\_initial\_vol\_m3) + prev(netFlow,0.0) < 0.0) => (current\_vol = 0.0);

guarantee "If the current tank 2 volume plus the tank 2 net flow is greater than empty and less than the tank 2 maximum volume then the tank 2 volume changes by the tank 2 net flow":

(prev(current\_vol, tank\_package.tank2\_initial\_vol\_m3) + prev(netFlow,0.0) <= tank\_package.max\_tank2\_vol\_m3) and (prev(current\_vol, tank\_package.tank2\_initial\_vol\_m3) + prev(netFlow,0.0) >= 0.0)

=> (current\_vol = prev(current\_vol, tank\_package.tank2\_initial\_vol\_m3) + prev(netFlow,0.0));

guarantee "Current height of tank 2 is equal or less than the max tank 2 height" :

Liq\_Height\_m <= max\_tank\_height;

guarantee "Current height of tank 2 is non negative" :

Liq\_Height\_m >= 0.0;

guarantee "The tank 2 current height is equal to the tank 2 current volume divided by the tank 2 cross sectional area" :

Liq\_Height\_m = current\_vol / tank\_package.tank2\_cross\_section\_area\_m2;

guarantee "The max tank 2 height is equal to the max tank 2 volume divided by tank 2 cross sectional area":

max\_tank\_height = tank\_package.max\_tank2\_vol\_m3 / tank\_package.tank2\_cross\_section\_area\_m2;

guarantee "The current tank 2 height is the current tank 2 volume divided by the tank 2 cross sectional area":

current\_height = (current\_vol / tank\_package.tank2\_cross\_section\_area\_m2);

guarantee "The output height of tank 2 should equal the calculated height of tank 2":

Liq\_Height\_m = current\_height;

\*\*};

end Tank\_2;

system implementation Tank\_2.UniformTankImpl

annex agree{\*\*

eq delta\_vol : real = prev(netFlow,0.0);

eq tank\_vol\_calc : real = ((tank\_package.tank2\_initial\_vol\_m3 -> pre(current\_vol)) + delta\_vol);

eq tank\_vol\_overflow\_underflow : real = if (tank\_vol\_calc > tank\_package.max\_tank2\_vol\_m3) then tank\_package.max\_tank2\_vol\_m3 else if (tank\_vol\_calc < 0.0) then 0.0 else tank\_vol\_calc;

assert(current\_vol = tank\_vol\_overflow\_underflow);

assert(max\_tank\_height = (tank\_package.max\_tank2\_vol\_m3 / tank\_package.tank2\_cross\_section\_area\_m2));

assert(current\_height = (current\_vol / tank\_package.tank2\_cross\_section\_area\_m2));

-- Output

assert(Liq\_Height\_m = current\_height);

\*\*};

end Tank\_2.UniformTankImpl;

end tank\_package;

==============================================================================

valve.aadl

==============================================================================

package valve\_package

public

with Base\_Types;

with config;

annex agree {\*\*

-- Tank 1 specifications ----------------------------

const t1\_max\_flow\_rate\_m3s : real = config.t1\_valve\_max\_flow\_rate\_m3s;

const t1\_zero\_flow\_rate\_m3s : real = config.t1\_valve\_zero\_flow\_rate\_m3s;

-----------------------------------------------------

-- Tank 2 specifications ----------------------------

const t2\_p\_max\_flow\_rate\_m3s : real = config.t2\_p\_valve\_max\_flow\_rate\_m3s;

const t2\_p\_zero\_flow\_rate\_m3s : real = config.t2\_valve\_zero\_flow\_rate\_m3s;

const t2\_e\_max\_flow\_rate\_m3s : real = config.t2\_e\_valve\_max\_flow\_rate\_m3s;

const t2\_e\_zero\_flow\_rate\_m3s : real = config.t2\_valve\_zero\_flow\_rate\_m3s;

-----------------------------------------------------

\*\*};

system T2\_Production\_Valve

features

Valve\_State: in data port Base\_Types::Boolean;

Flow\_Out: out data port Base\_Types::Float;

annex agree {\*\*

guarantee "If the production valve is open then the flow through the valve is the max production valve flow rate" :

Valve\_State => (Flow\_Out = valve\_package.t2\_p\_max\_flow\_rate\_m3s);

guarantee "If the production valve is closed then the flow rate through the production valve is 0.0" :

(not Valve\_State) => (Flow\_Out = valve\_package.t2\_p\_zero\_flow\_rate\_m3s);

guarantee "If the Flow is the max production valve rate, then the valve is receiving an OPEN signal" :

(Flow\_Out = valve\_package.t2\_p\_max\_flow\_rate\_m3s) => Valve\_State;

guarantee "If the Flow is 0.0, then the production valve is receiving a CLOSED signal" :

(Flow\_Out = valve\_package.t2\_p\_zero\_flow\_rate\_m3s) => not Valve\_State;

\*\*};

end T2\_Production\_Valve;

system implementation T2\_Production\_Valve.Impl

annex agree {\*\*

eq flow\_rate : real = if (Valve\_State) then valve\_package.t2\_p\_max\_flow\_rate\_m3s else valve\_package.t2\_p\_zero\_flow\_rate\_m3s;

-- This is to assert that the output acts as it should

assert(Flow\_Out = flow\_rate);

\*\*};

end T2\_Production\_Valve.Impl;

system T2\_Emergency\_Valve

features

Valve\_State: in data port Base\_Types::Boolean;

Flow\_Out: out data port Base\_Types::Float;

annex agree {\*\*

guarantee "If the emergency valve is open then the flow through the valve is the max emergency valve flow rate" :

Valve\_State => (Flow\_Out = valve\_package.t2\_e\_max\_flow\_rate\_m3s);

guarantee "If the emergency valve is closed then the flow rate through the emergency valve is 0.0" :

(not Valve\_State) => (Flow\_Out = valve\_package.t2\_e\_zero\_flow\_rate\_m3s);

guarantee "If the Flow is the max emergency valve rate, then the emergency valve is receiving an OPEN signal" :

(Flow\_Out = valve\_package.t2\_e\_max\_flow\_rate\_m3s) => Valve\_State;

guarantee "If the Flow is 0.0, then the emergency valve is receiving a CLOSED signal" :

(Flow\_Out = valve\_package.t2\_e\_zero\_flow\_rate\_m3s) => not Valve\_State;

\*\*};

end T2\_Emergency\_Valve;

system implementation T2\_Emergency\_Valve.Impl

annex agree {\*\*

eq flow\_rate : real = if (Valve\_State) then valve\_package.t2\_e\_max\_flow\_rate\_m3s else valve\_package.t2\_e\_zero\_flow\_rate\_m3s;

-- This is to assert that the output acts as it should

assert(Flow\_Out = flow\_rate);

\*\*};

end T2\_Emergency\_Valve.Impl;

system T1\_Valve

features

Valve\_State: in data port Base\_Types::Boolean;

Flow\_Out: out data port Base\_Types::Float;

annex agree {\*\*

guarantee "If the valve is open then the flow through the valve is the max valve flow rate" :

Valve\_State => (Flow\_Out = valve\_package.t1\_max\_flow\_rate\_m3s);

guarantee "If the valve is closed then the flow rate through the valve is 0.0" :

(not Valve\_State) => (Flow\_Out = valve\_package.t1\_zero\_flow\_rate\_m3s);

guarantee "If the Flow is the max valve rate, then the valve is receiving an OPEN signal" :

(Flow\_Out = valve\_package.t1\_max\_flow\_rate\_m3s) => Valve\_State;

guarantee "If the Flow is 0.0, then the valve is receiving a CLOSED signal" :

(Flow\_Out = valve\_package.t1\_zero\_flow\_rate\_m3s) => not Valve\_State;

\*\*};

end T1\_Valve;

system implementation T1\_Valve.Impl

annex agree {\*\*

eq flow\_rate : real = if (Valve\_State) then valve\_package.t1\_max\_flow\_rate\_m3s else valve\_package.t1\_zero\_flow\_rate\_m3s;

-- This is to assert that the output acts as it should

assert(Flow\_Out = flow\_rate);

\*\*};

end T1\_Valve.Impl;

end valve\_package;

# Appendix D

This is the source code for the AGREE primitive representation of the Basic Control Tanks System

==============================================================================

binary\_switch.aadl

==============================================================================

package binary\_switch\_primitive\_pkg

public

with Base\_Types;

system binary\_switch

features

true\_case : in data port Base\_Types::Float; -- u1

false\_case : in data port Base\_Types::Float; -- u2

control : in data port Base\_Types::Boolean;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be equal to the true\_case when the control is true":

(control = true) => (output = true\_case);

guarantee "The output shall be equal to the false\_case when the control is false":

(control = false) => (output = false\_case);

\*\*};

end binary\_switch;

system implementation binary\_switch.impl

annex agree{\*\*

eq out : real = if (control = true) then true\_case else false\_case;

assert(output = out);

\*\*};

end binary\_switch.impl;

system boolean\_binary\_switch

features

true\_case : in data port Base\_Types::Boolean; -- u1

false\_case : in data port Base\_Types::Boolean; -- u2

control : in data port Base\_Types::Boolean;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall be equal to the true\_case when the control is true":

(control = true) => (output = true\_case);

guarantee "The output shall be equal to the false\_case when the control is false":

(control = false) => (output = false\_case);

\*\*};

end boolean\_binary\_switch;

system implementation boolean\_binary\_switch.impl

annex agree{\*\*

eq out : bool = if (control = true) then true\_case else false\_case;

assert(output = out);

\*\*};

end boolean\_binary\_switch.impl;

end binary\_switch\_primitive\_pkg;

==============================================================================

constant.aadl

==============================================================================

package constant\_primitive\_pkg

public

with Base\_Types;

system constant\_1

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 1.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_1;

system implementation constant\_1.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_1.impl;

system boolean\_true

features

output : out data port Base\_Types::Boolean;

annex agree{\*\*

const value : bool = true;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end boolean\_true;

system implementation boolean\_true.impl

annex agree{\*\*

eq constant\_value : bool = value;

assert(output = constant\_value);

\*\*};

end boolean\_true.impl;

system constant\_0

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 0.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_0;

system implementation constant\_0.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_0.impl;

system boolean\_false

features

output : out data port Base\_Types::Boolean;

annex agree{\*\*

const value : bool = false;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end boolean\_false;

system implementation boolean\_false.impl

annex agree{\*\*

eq constant\_value : bool = value;

assert(output = constant\_value);

\*\*};

end boolean\_false.impl;

system constant\_5

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 5.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_5;

system implementation constant\_5.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_5.impl;

system constant\_7

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 7.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_7;

system implementation constant\_7.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_7.impl;

system constant\_8

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 8.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_8;

system implementation constant\_8.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_8.impl;

system constant\_2

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 2.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_2;

system implementation constant\_2.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_2.impl;

system constant\_3

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 3.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_3;

system implementation constant\_3.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_3.impl;

system constant\_0\_2

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 0.2;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_0\_2;

system implementation constant\_0\_2.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_0\_2.impl;

system constant\_0\_1

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 0.1;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_0\_1;

system implementation constant\_0\_1.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_0\_1.impl;

system constant\_0\_05

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = 0.05;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_0\_05;

system implementation constant\_0\_05.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_0\_05.impl;

system constant\_neg1

features

output : out data port Base\_Types::Float;

annex agree{\*\*

const value : real = -1.0;

guarantee "The output shall be the constant value":

output = value;

guarantee "the output shall never change":

true -> (output = pre(output));

\*\*};

end constant\_neg1;

system implementation constant\_neg1.impl

annex agree{\*\*

eq constant\_value : real = value;

assert(output = constant\_value);

\*\*};

end constant\_neg1.impl;

end constant\_primitive\_pkg;

==============================================================================

delay.aadl

==============================================================================

package delay\_primitive\_pkg

public

with Base\_Types;

system delay

features

input : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The current output shall be the previous input":

output = prev(input,0.0);

guarantee "the output is initially 0":

std.Initial(output = 0.0);

\*\*};

end delay;

system implementation delay.impl

annex agree{\*\*

eq delayed\_input : real = prev(input,0.0);

assert(output = delayed\_input);

\*\*};

end delay.impl;

system delay\_with\_initial

features

input : in data port Base\_Types::Float;

initial\_value : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The current output shall be the previous input":

output = prev(input,initial\_value);

guarantee "the output is initially the initial\_value":

std.Initial(output = initial\_value);

\*\*};

end delay\_with\_initial;

system implementation delay\_with\_initial.impl

annex agree{\*\*

eq delayed\_input : real = prev(input,initial\_value);

assert(output = delayed\_input);

\*\*};

end delay\_with\_initial.impl;

end delay\_primitive\_pkg;

==============================================================================

initializer.aadl

==============================================================================

package initializer\_primitive\_pkg

public

with Base\_Types;

with std;

with delay\_primitive\_pkg;

with binary\_switch\_primitive\_pkg;

with constant\_primitive\_pkg;

with relational\_primitive\_pkg;

system initializer

features

initial\_value\_in : in data port Base\_Types::Float;

initial\_value\_out : out data port Base\_Types::Float;

annex agree {\*\*

guarantee "Initially, the output is equal to the input":

std.Initial(initial\_value\_out = initial\_value\_in);

guarantee "after the first time step, the output is equal to zero":

std.After1(initial\_value\_out = 0.0);

\*\*};

end initializer;

system implementation initializer.impl

subcomponents

switch : system binary\_switch\_primitive\_pkg::binary\_switch.impl;

one : system constant\_primitive\_pkg::constant\_1.impl;

zero : system constant\_primitive\_pkg::constant\_0.impl;

compare\_to\_one : system relational\_primitive\_pkg::compare\_1.impl;

delay : system delay\_primitive\_pkg::delay\_with\_initial.impl;

connections

-- Setup Delay

zero\_to\_delay\_input : port zero.output -> delay.input {Communication\_Properties::Timing => immediate;};

one\_to\_delay\_initial : port one.output -> delay.initial\_value {Communication\_Properties::Timing => immediate;};

-- Setup Compare

delay\_to\_compare : port delay.output -> compare\_to\_one.input {Communication\_Properties::Timing => immediate;};

-- Setup Switch

input\_to\_switch : port initial\_value\_in -> switch.true\_case {Communication\_Properties::Timing => immediate;};

zero\_to\_switch : port zero.output -> switch.false\_case {Communication\_Properties::Timing => immediate;};

compare\_to\_switch : port compare\_to\_one.output -> switch.control {Communication\_Properties::Timing => immediate;};

-- Output

switch\_to\_output : port switch.output -> initial\_value\_out {Communication\_Properties::Timing => immediate;};

end initializer.impl;

end initializer\_primitive\_pkg;

==============================================================================

limiter.aadl

==============================================================================

package limiter\_primitive\_pkg

public

with Base\_Types;

system limiter

features

up : in data port Base\_Types::Float;

input : in data port Base\_Types::Float;

lo : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

-- Assumptions

assume "upper limit is greater than lower limit":

up > lo;

-- Guarantees

guarantee "If the input is greater than the upper limit then the output is eqaul to the upper limit":

(input > up) => (output = up);

guarantee "If the input is less than the lower limit then the output is eqaul to the lower limit":

(input < lo) => (output = lo);

guarantee "If the input is between the upper and lower limits then the output is eqaul to the input":

(input <= up and input >= lo) => (output = input);

\*\*};

end limiter;

system implementation limiter.impl

annex agree{\*\*

eq out : real = if input > up then up else if input < lo then lo else input;

assert(output = out);

\*\*};

end limiter.impl;

end limiter\_primitive\_pkg;

==============================================================================

not.aadl

==============================================================================

package not\_primitive\_pkg

public

with Base\_Types;

system not

features

input : in data port Base\_Types::Boolean;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall be false if the input is true":

(input = true) => (output = false);

guarantee "The output shall be true if the input is false":

(input = false) => (output = true);

\*\*};

end not;

system implementation not.impl

annex agree{\*\*

eq not\_var : bool = not input;

assert(output = not\_var);

\*\*};

end not.impl;

system not\_r

features

input : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be false if the input is true":

(input = 1.0) => (output = 0.0);

guarantee "The output shall be true if the input is false":

(input = 0.0) => (output = 1.0);

\*\*};

end not\_r;

system implementation not\_r.impl

annex agree{\*\*

eq not\_var : real = if input = 1.0 then 0.0 else 1.0;

assert(output = not\_var);

\*\*};

end not\_r.impl;

end not\_primitive\_pkg;

==============================================================================

relational.aadl

==============================================================================

package relational\_primitive\_pkg

public

with Base\_Types;

system relational\_ge

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is greater or equal to input2":

output = (input1 >= input2);

\*\*};

end relational\_ge;

system implementation relational\_ge.impl

annex agree{\*\*

eq greater\_e : bool = (input1 >= input2);

assert(output = greater\_e);

\*\*};

end relational\_ge.impl;

system relational\_g

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is greater than input2":

output = (input1 > input2);

\*\*};

end relational\_g;

system implementation relational\_g.impl

annex agree{\*\*

eq greater : bool = (input1 > input2);

assert(output = greater);

\*\*};

end relational\_g.impl;

system relational\_le

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is less than or equal to input2":

output = (input1 <= input2);

\*\*};

end relational\_le;

system implementation relational\_le.impl

annex agree{\*\*

eq less\_e : bool = (input1 <= input2);

assert(output = less\_e);

\*\*};

end relational\_le.impl;

system relational\_l

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is less than input2":

output = (input1 < input2);

\*\*};

end relational\_l;

system implementation relational\_l.impl

annex agree{\*\*

eq less : bool = (input1 < input2);

assert(output = less);

\*\*};

end relational\_l.impl;

system relational\_e

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is equal to input2":

output = (input1 = input2);

\*\*};

end relational\_e;

system implementation relational\_e.impl

annex agree{\*\*

eq e : bool = (input1 = input2);

assert(output = e);

\*\*};

end relational\_e.impl;

system relational\_ne

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall true if input1 is not equal to input2":

output = (input1 <> input2);

\*\*};

end relational\_ne;

system implementation relational\_ne.impl

annex agree{\*\*

eq ne : bool = (input1 <> input2);

assert(output = ne);

\*\*};

end relational\_ne.impl;

system compare\_1

features

input : in data port Base\_Types::Float;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

const val : real = 1.0;

guarantee "The output shall true if input1 is equal to input2":

output = (input = val);

\*\*};

end compare\_1;

system implementation compare\_1.impl

annex agree{\*\*

eq e : bool = (input = val);

assert(output = e);

\*\*};

end compare\_1.impl;

end relational\_primitive\_pkg;

==============================================================================

two\_input\_and.aadl

==============================================================================

package two\_input\_and\_primitive\_pkg

public

with Base\_Types;

system two\_input\_and

features

input1 : in data port Base\_Types::Boolean;

input2 : in data port Base\_Types::Boolean;

output : out data port Base\_Types::Boolean;

annex agree{\*\*

guarantee "The output shall be false if input1 is false":

(input1 = false) => (output = false);

guarantee "The output shall be false if input2 is false":

(input2 = false) => (output = false);

guarantee "The output shall be true if input1 and input2 are both true":

(input1 and input2) => (output = true);

\*\*};

end two\_input\_and;

system implementation two\_input\_and.impl

annex agree{\*\*

eq and\_var : bool = input1 and input2;

assert(output = and\_var);

\*\*};

end two\_input\_and.impl;

end two\_input\_and\_primitive\_pkg;

==============================================================================

two\_input\_diff.aadl

==============================================================================

package two\_input\_diff\_primitive\_pkg

public

with Base\_Types;

system two\_input\_diff

features

minuend : in data port Base\_Types::Float; -- u1

subtrahend : in data port Base\_Types::Float; -- u2

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be the difference between the minuend and subtrahend":

output = minuend - subtrahend;

\*\*};

end two\_input\_diff;

system implementation two\_input\_diff.impl

annex agree{\*\*

eq diff : real = minuend - subtrahend;

assert(output = diff);

\*\*};

end two\_input\_diff.impl;

end two\_input\_diff\_primitive\_pkg;

==============================================================================

two\_input\_division.aadl

==============================================================================

package two\_input\_division\_primitive\_pkg

public

with Base\_Types;

system two\_input\_division

features

dividend : in data port Base\_Types::Float; -- u1

divisor : in data port Base\_Types::Float; -- u2

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be the division of the two inputs":

output = dividend / divisor;

\*\*};

end two\_input\_division;

system implementation two\_input\_division.impl

annex agree{\*\*

eq quotient : real = dividend / divisor;

assert(output = quotient);

\*\*};

end two\_input\_division.impl;

end two\_input\_division\_primitive\_pkg;

==============================================================================

two\_input\_product.aadl

==============================================================================

package two\_input\_product\_primitive\_pkg

public

with Base\_Types;

system two\_input\_product

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be the product of the two inputs":

output = input1 \* input2;

\*\*};

end two\_input\_product;

system implementation two\_input\_product.impl

annex agree{\*\*

eq product : real = input1 \* input2;

assert(output = product);

\*\*};

end two\_input\_product.impl;

end two\_input\_product\_primitive\_pkg;

==============================================================================

two\_input\_sum.aadl

==============================================================================

package two\_input\_sum\_primitive\_pkg

public

with Base\_Types;

system two\_input\_sum

features

input1 : in data port Base\_Types::Float;

input2 : in data port Base\_Types::Float;

output : out data port Base\_Types::Float;

annex agree{\*\*

guarantee "The output shall be the sum of the two inputs":

output = input1 + input2;

\*\*};

end two\_input\_sum;

system implementation two\_input\_sum.impl

annex agree{\*\*

eq sum : real = input1 + input2;

assert(output = sum);

\*\*};

end two\_input\_sum.impl;

end two\_input\_sum\_primitive\_pkg;

# Appendix E

This is the Simulink model for the functional representation of the Basic Control Tanks System



Figure 1: Top view of the Functional Simulink Model

This is the Simulink model for the primitive representation of the Basic Control Tanks System

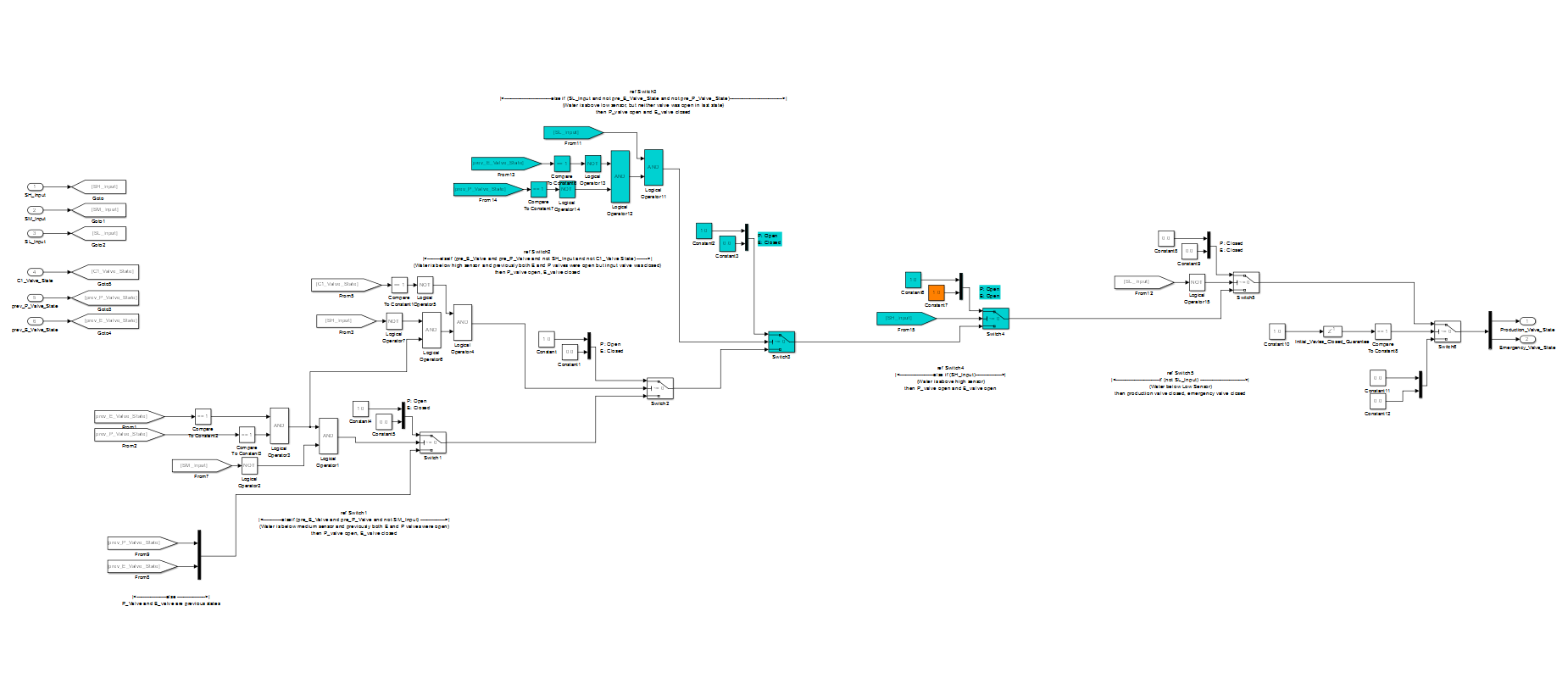


Figure 2: Simulink model of the Primitive Model

# Appendix F

Description of the SLDV representation of the primitive Basic Control Tanks System

Coupled Tanks Simulink Files

Kerianne Gross

March 12, 2014

This document provides an introductory overview of the Coupled Tanks Simulink Model and Verification. It is assumed the reader has a basic understanding of MATLAB, Simulink, AADL, AGREE, and formal methods analysis, but may not have knowledge of Simulink Design Verifier.

Contents

[List of Figures 4](#_Toc414014566)

[Comments on the Model 5](#_Toc414014567)

[Before you run the files 5](#_Toc414014568)

[File Structure 6](#_Toc414014569)

[Simulation of System Level Model 7](#_Toc414014570)

[Controller 8](#_Toc414014571)

[Tank 1 Controller 8](#_Toc414014572)

[Tank 2 Controller 9](#_Toc414014573)

[Environment 10](#_Toc414014574)

[Actuators 10](#_Toc414014575)

[Tank 1 11](#_Toc414014576)

[Tank 2 11](#_Toc414014577)

[Sensors 11](#_Toc414014578)

[Assumptions 13](#_Toc414014579)

[Properties 13](#_Toc414014580)

[Persistent Variables 13](#_Toc414014581)

[Verification Files 14](#_Toc414014582)

[Properties and Assumptions used to Analyze the Subsystems 15](#_Toc414014583)

[Actuator 15](#_Toc414014584)

[Assumptions 15](#_Toc414014585)

[Properties/Guarantees (Proof Objectives) 15](#_Toc414014586)

[Controller for Tank 1 15](#_Toc414014587)

[Assumptions 15](#_Toc414014588)

[Properties/Guarantees (Proof Objectives) 15](#_Toc414014589)

[Controller for Tank 2 15](#_Toc414014590)

[Assumptions 15](#_Toc414014591)

[Properties/Guarantees (Proof Objectives) 16](#_Toc414014592)

[Sensor 16](#_Toc414014593)

[Assumptions 16](#_Toc414014594)

[Properties/Guarantees (Proof Objectives) 16](#_Toc414014595)

[Tank 1 16](#_Toc414014596)

[Assumptions 16](#_Toc414014597)

[Properties/Guarantees (Proof Objectives) 17](#_Toc414014598)

[Tank 2 17](#_Toc414014599)

[Assumptions 17](#_Toc414014600)

[Properties/Guarantees (Proof Objectives) 17](#_Toc414014601)

[Verification 18](#_Toc414014602)

# List of Figures

Figure 1: Menus to “Add Selected Folders and Subfolders” to the Path 5

Figure 2: Location of constants file in the folder 5

Figure 3: File structure of the model 6

Figure 4: System Level Simulation of the Model 7

Figure 5: Plot of the liquid height in Tank 1 (Yellow) and liquid height in Tank 2 (magenta) versus time 7

Figure 6: Controller Model 8

Figure 7: Tank 1 Controller Model 8

Figure 8: Tank 2 Controller Model 9

Figure 9: Environment Level Model 10

Figure 10: Actuator Model 10

Figure 11: Tank 1 Model 11

Figure 12: Tank 2 Model 11

Figure 13: Sensor Model 11

Figure 14: Example of Property File (from the Sensor Property File) 12

Figure 15: Persistent Variable Example (from the Tank 1 Controller Property File) 13

Figure 16: Location of Simulink Design Verifier Blocks in Library Browser 14

Figure 17: Verification File Example (from Sensor verification file) 14

Figure 18: Design Verifier Menu Navigation for Analysis 18

Figure 19: Simulink Design Verifier Log Window 18

# Comments on the Model

* Files were created in Matlab 2012b
* In order to run the verification files, you must have Simulink Design Verifier installed
* The other files do not require Simulink Design Verifier

# Before you run the files

* Right click on the folder in the Matlab Current Folder Window, and follow the menus to “Add to Path” and “Selected Folders and Subfolders”

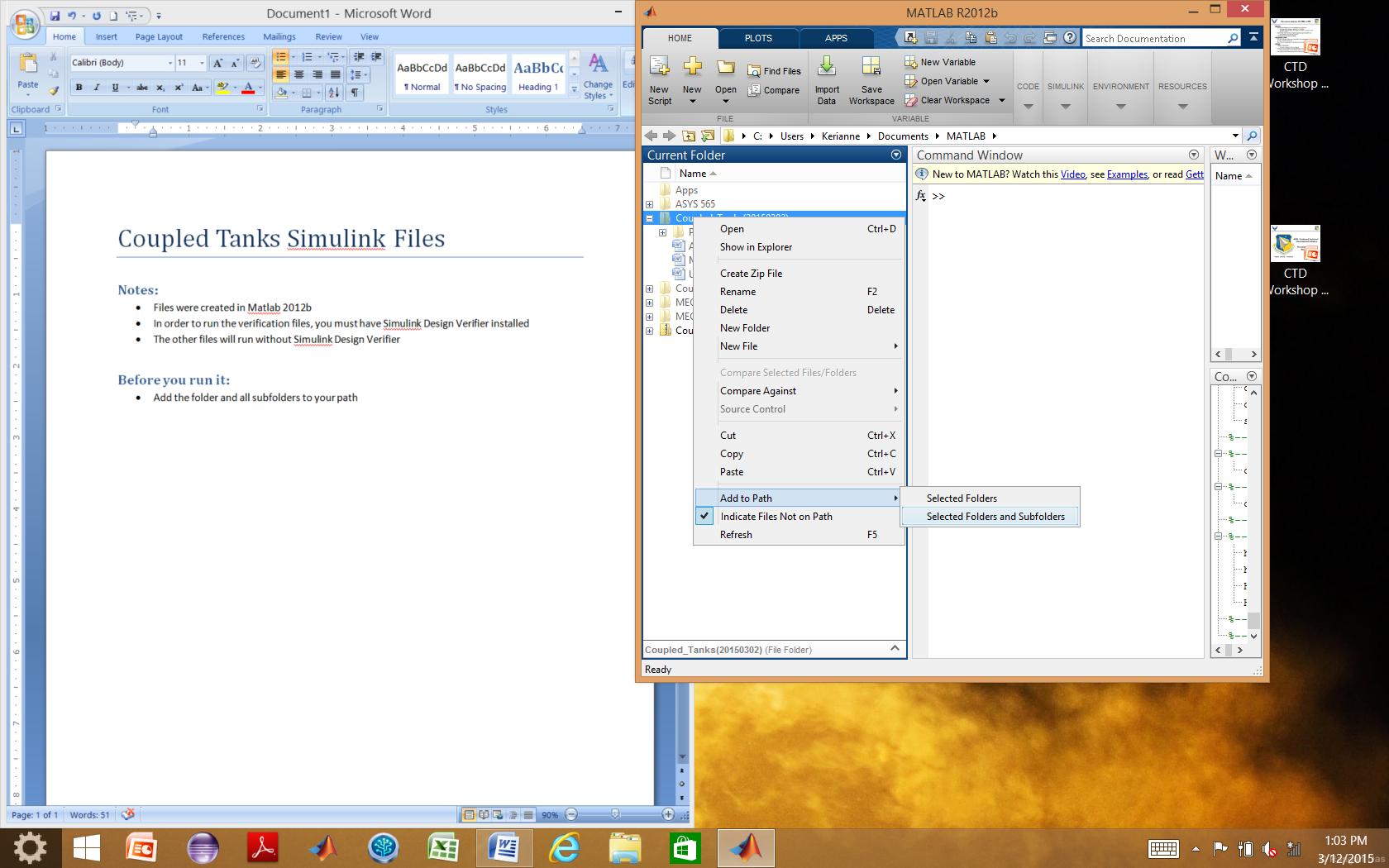


Figure 1: Menus to “Add Selected Folders and Subfolders” to the Path

* Run the “constants.m” file prior to any of the other files

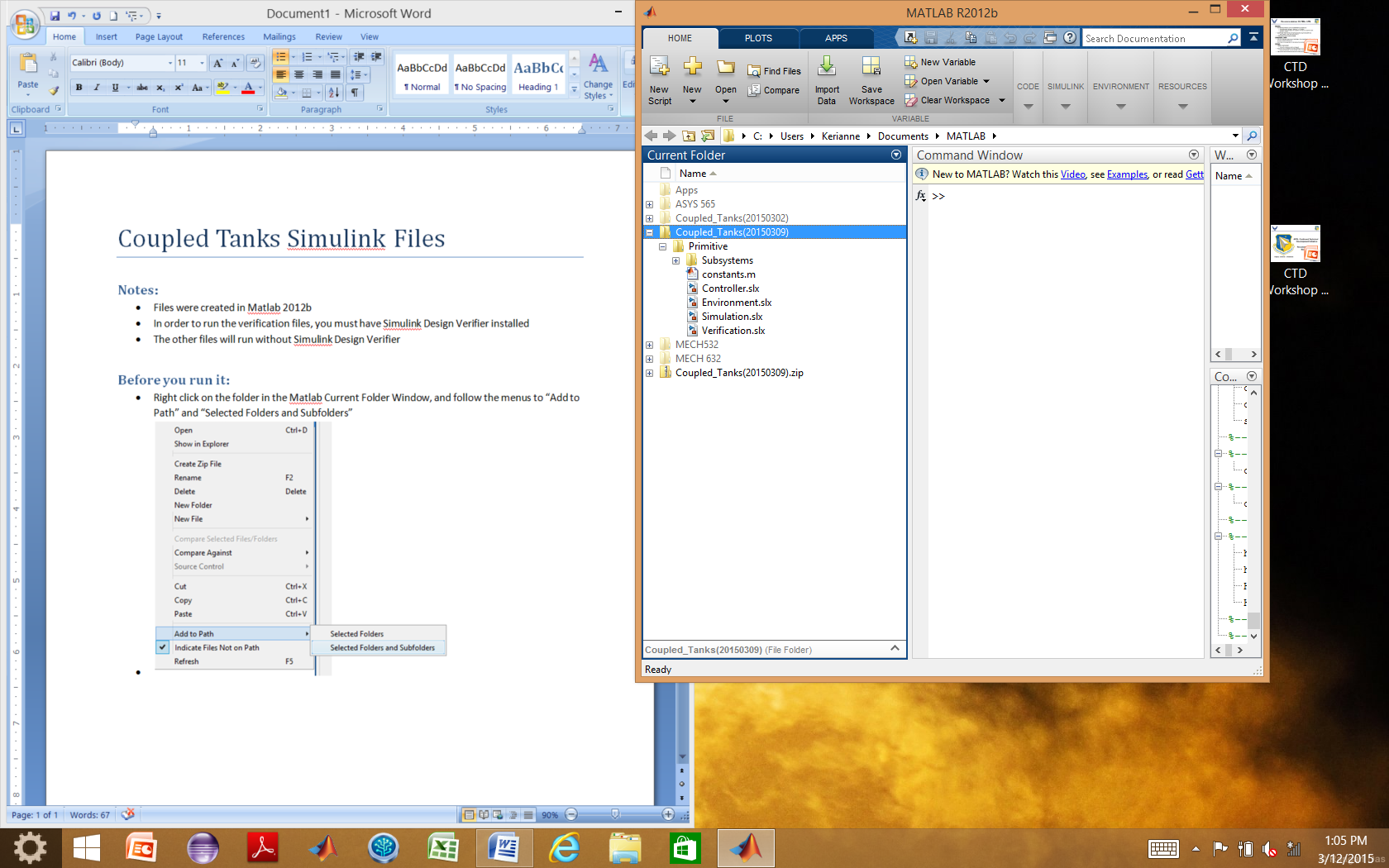


Figure 2: Location of constants file in the folder

# File Structure

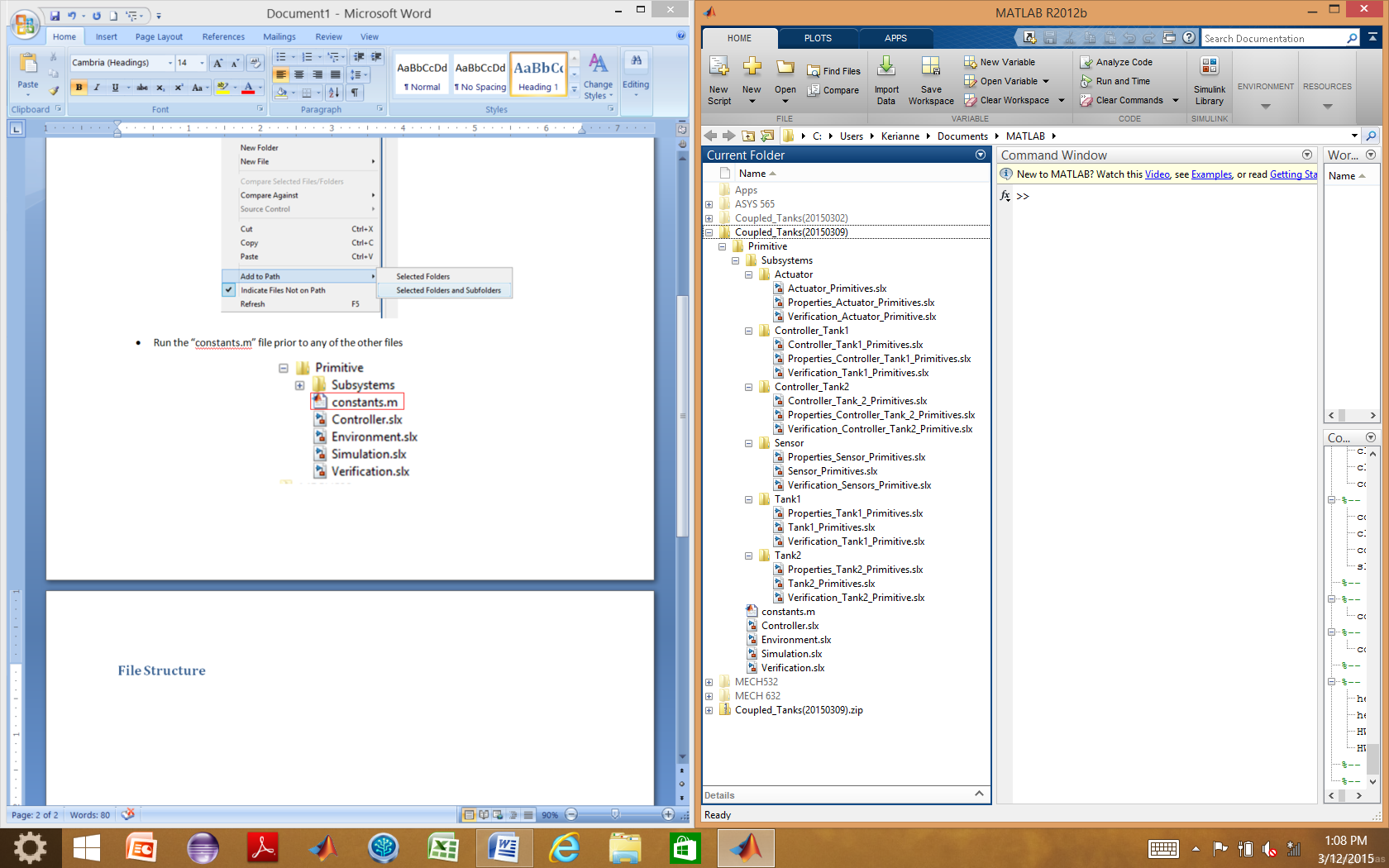


Figure 3: File structure of the model

* System level files are in the main folder, “Primitive” because this model is implemented using includes all of the formally specified primitives
* Subsystem level files are in the subsystem folders
* The **Simulation.slx** is a system level file that references the **Controller.slx** and **Environment.slx**
* The **Controller.slx** file references:
  + Controller\_Tank1\_Primitives.slx
  + Controller\_Tank2\_Primitives.slx
* The **Environment.slx** file references:
  + Actuator\_Primitives.slx
  + Sensor\_Primitives.slx
  + Tank1\_Primitives.slx
  + Tank2\_Primitives.slx
* The **Verification.slx** is a system level file that references **Simulation.slx** to do analysis on the system
* Each of the **Properties files of the subsystems** contains Properties that match the Guarantees and Assumptions that match Assumptions developed using AGREE in the AADL architecture phase
* Each of the subsystem Verification files references that subsystem model and the subsystem properties

# Simulation of System Level Model

* As stated in the File Structure Section, **Simulation.slx** is a system level file that references the **Controller.slx** and **Environment.slx**
* It ports the signals appropriately between the controller and environment and includes a scope to view the change in the height of the liquid in each tank
* A unit delay is placed between the signals from the environment to the controller to prevent an algebraic loop by accounting for the delay in time steps in the simulation
* Simulation.slx is shown in Figure 4

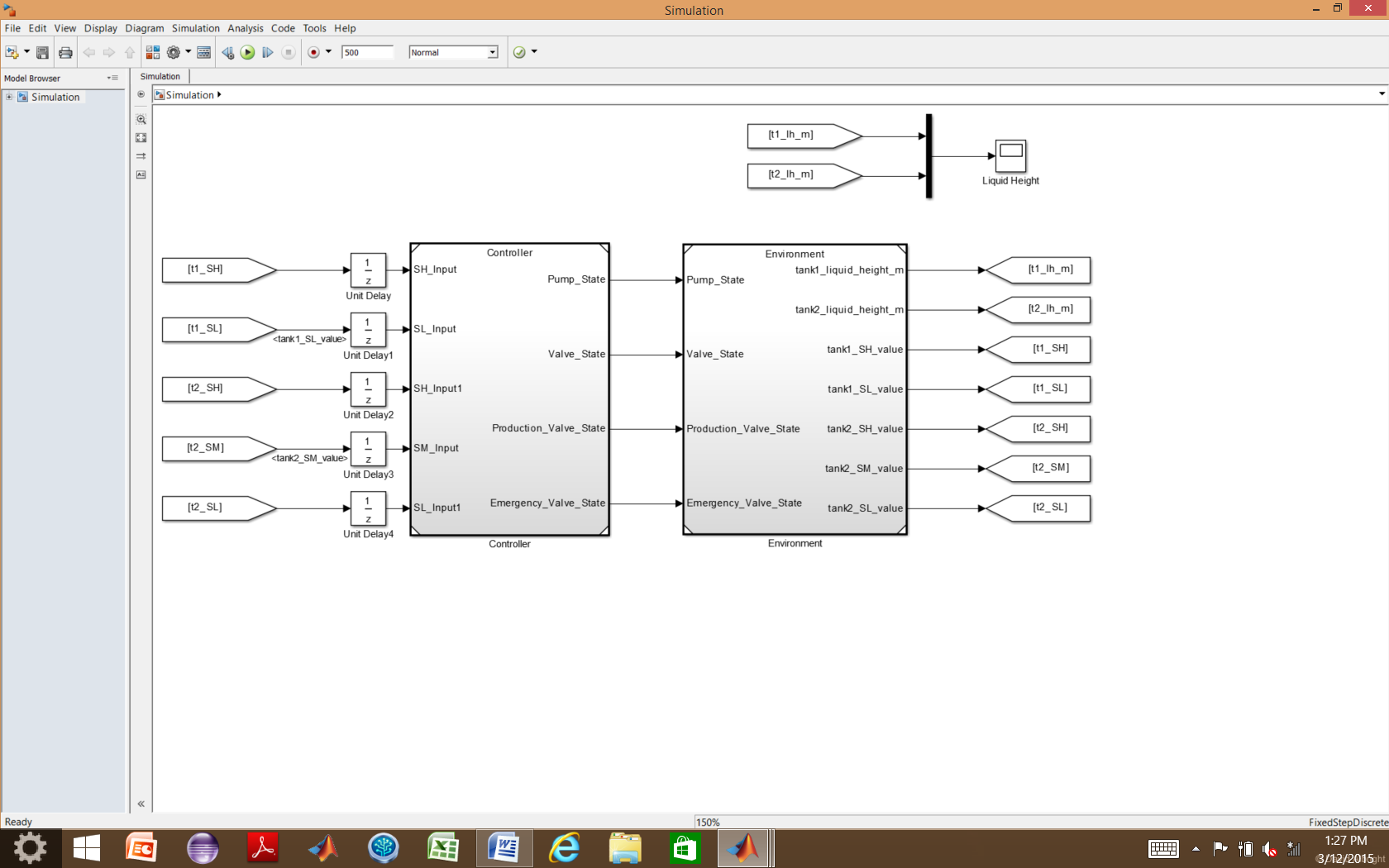


Figure 4: System Level Simulation of the Model



Figure 5: Plot of the liquid height in Tank 1 (Yellow) and liquid height in Tank 2 (magenta) versus time

# Controller

* The **Controller.slx** file references **Controller\_Tank1\_Primitives.slx** and **Controller\_Tank2\_Primitives.slx** and appropriately routes signals between the two
* Sensor Inputs are from the environment, but do not actually reference anything at this level

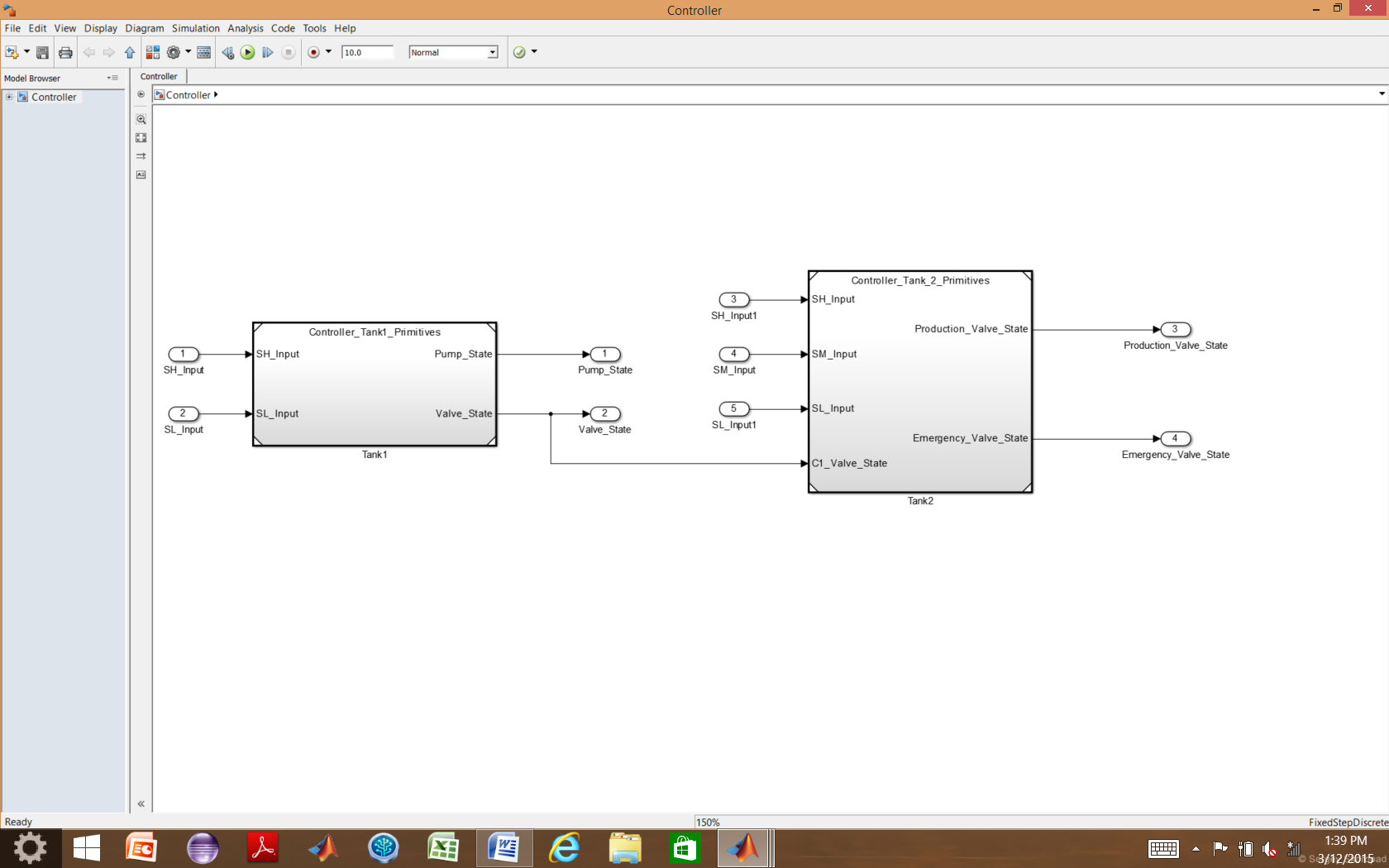


Figure 6: Controller Model

## Tank 1 Controller

The tank 1 controller implemented with Simulink primitives is shown in Figure 7. Unit delays are utilized to create the “previous state” of the pump and valve, used by the controller.

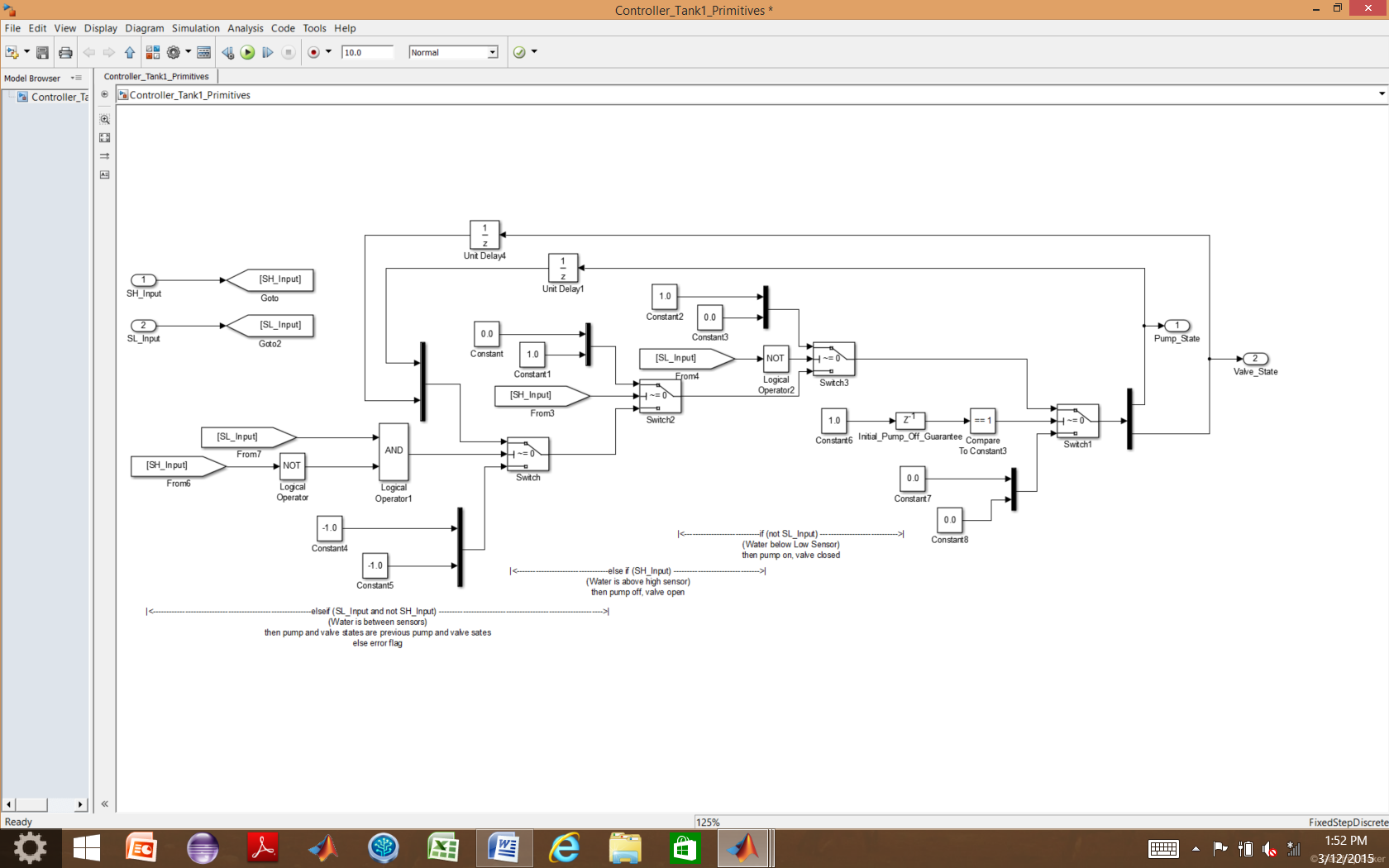
****

Figure 7: Tank 1 Controller Model

## Tank 2 Controller

The tank 2 controller implemented with Simulink primitives is shown in Figure 8. Delay blocks are used to compute the “previous state” of the Emergency and Production valve for use by the controller in the next time step.

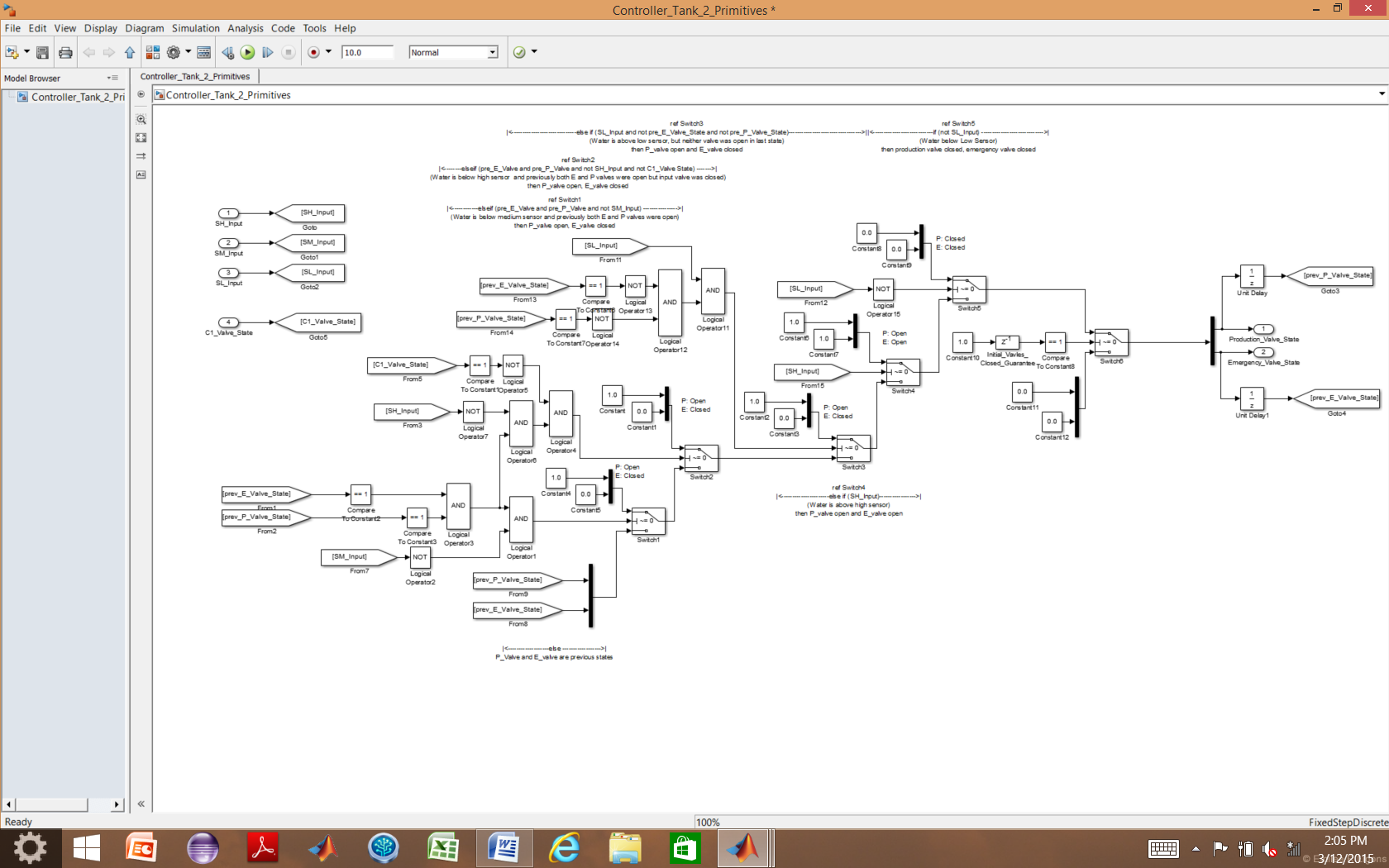


Figure 8: Tank 2 Controller Model

# Environment

The **Environment.slx** file references:

* Actuator\_Primitives.slx four times to model the output flow rate of the Pump, Valve, Production Valve, and Emergency Valve based on the signals each receives from the controller, the time increment, and the flow rate of each component (listed in the constants.m file)
* Tank1\_Primitives.slx to model the liquid height of Tank1 based on the flow rates of the Pump and Valve
* Tank2\_Primitives.slx to model the liquid height of Tank2 based on the flow rates of the Valve, Production Valve, and Emergency Valve
* Sensor\_Primitives.slx five times to model the each of the five sensor reactions (true or false) to the current height of the liquid based on its location (defined by the constants.m file) in its respective tank
* Unit Delay blocks are incorporated between the Tanks and the Sensors

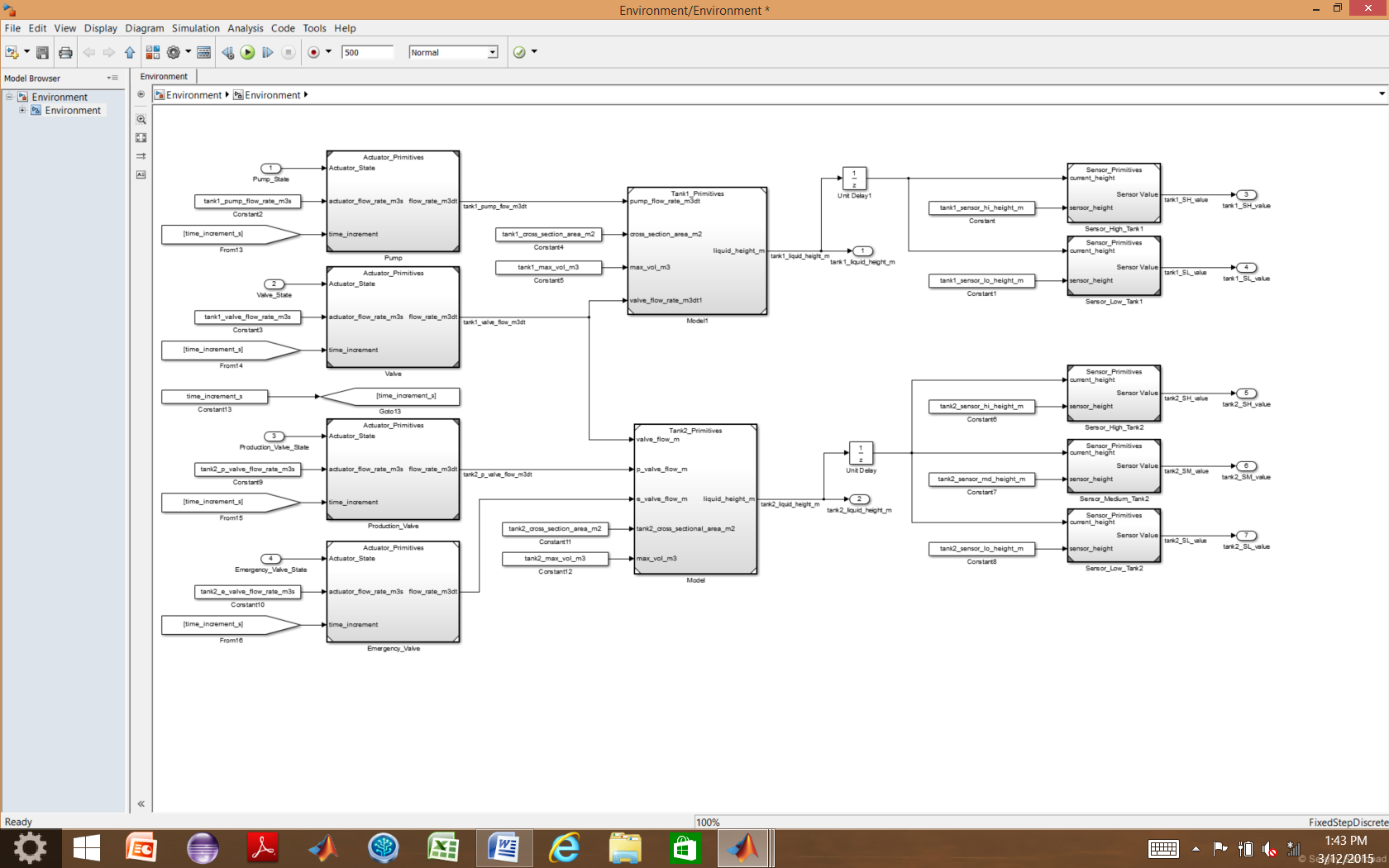


Figure 9: Environment Level Model

## Actuators

The same actuator model, shown in Figure 10, is used to model the performance of the Pump, Valve, Production Valve, and Emergency Valve.

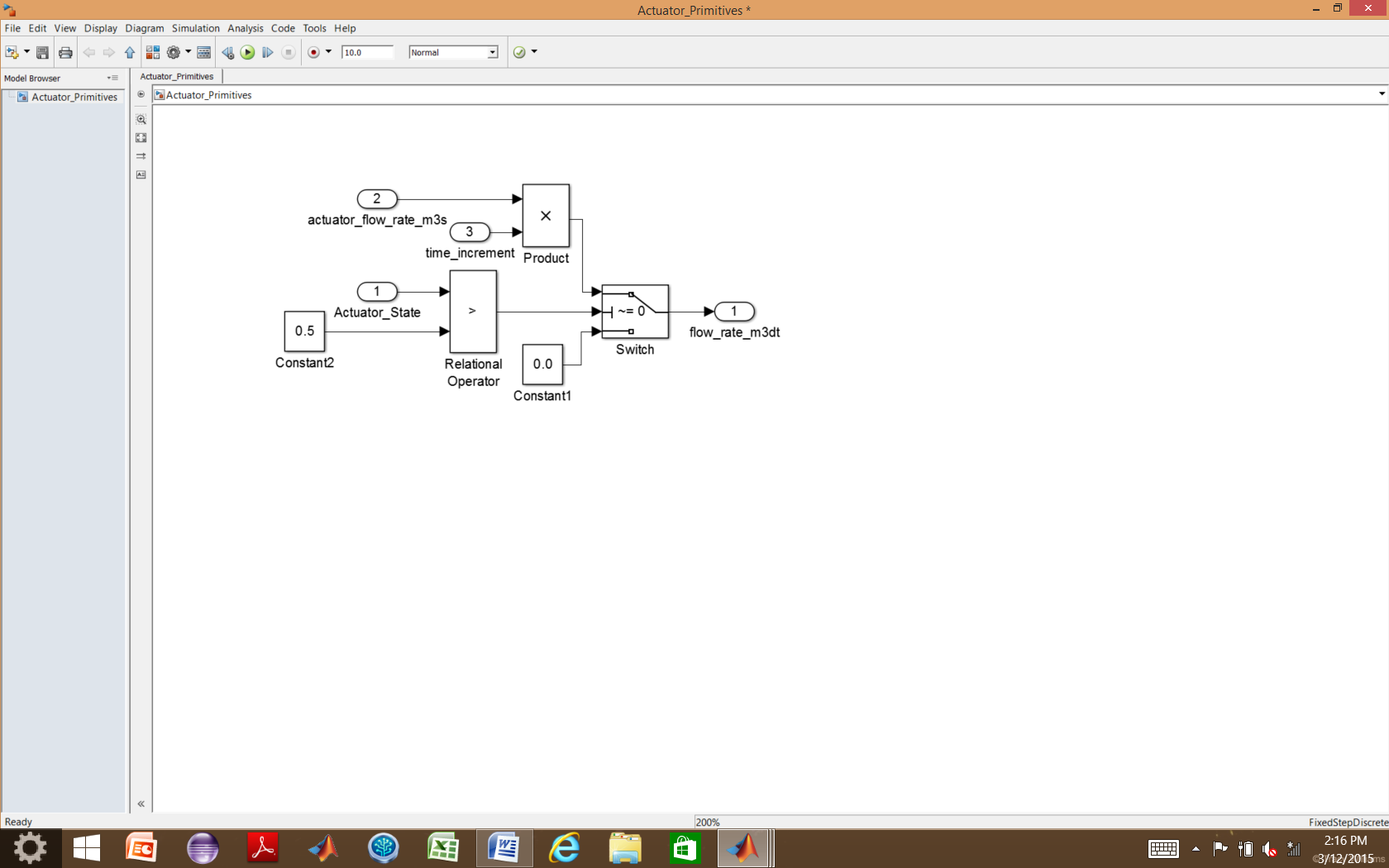


Figure 10: Actuator Model

## Tank 1

The model used to simulate Tank 1 is shown in Figure 11. A unit delay is used to calculate the “previous height” used in the next time step’s calculations.

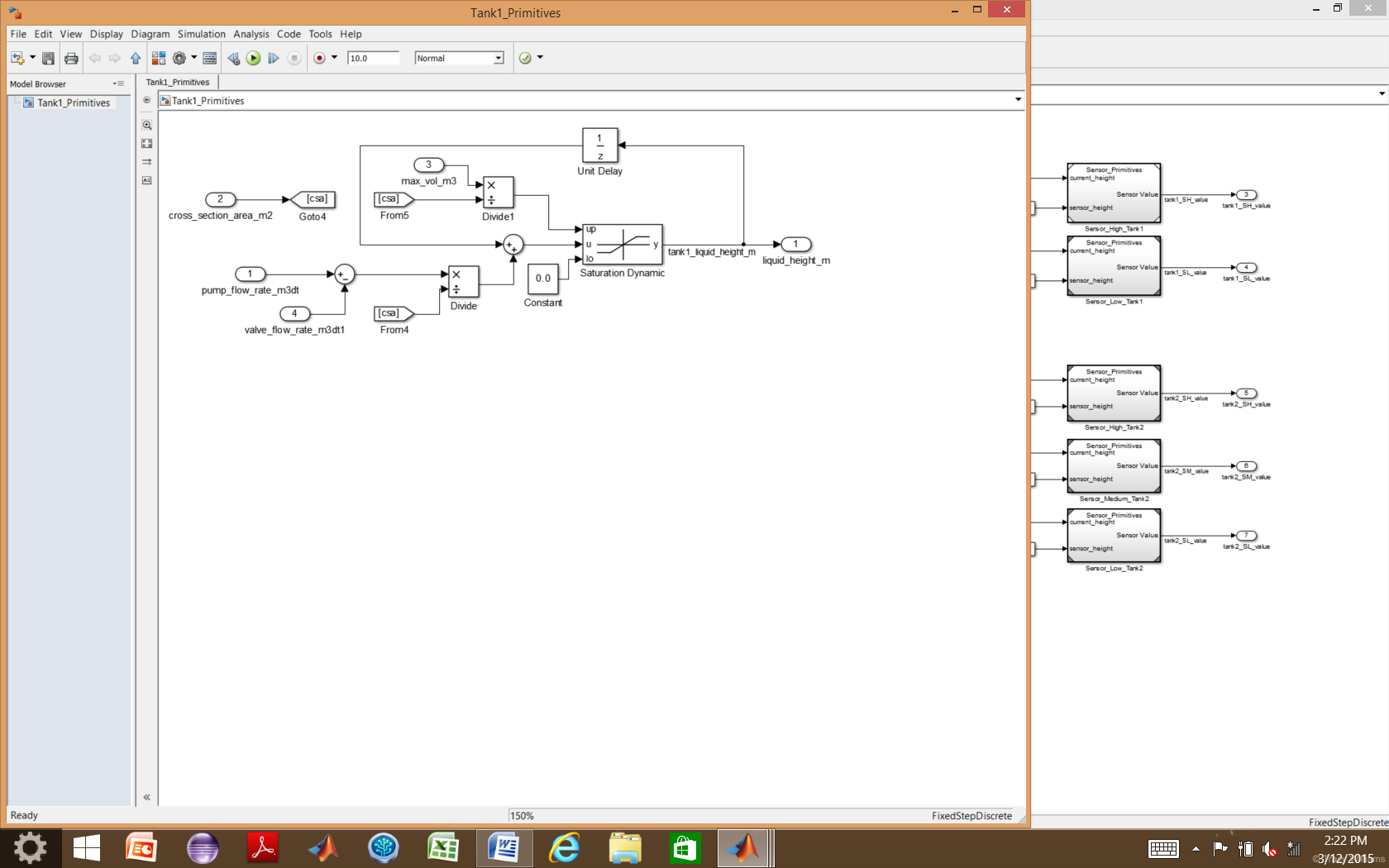


Figure 11: Tank 1 Model

## Tank 2

The model used to simulate Tank 2 is shown in Figure 12. A unit delay is used to calculate the “previous height” used in the next time step’s calculations.

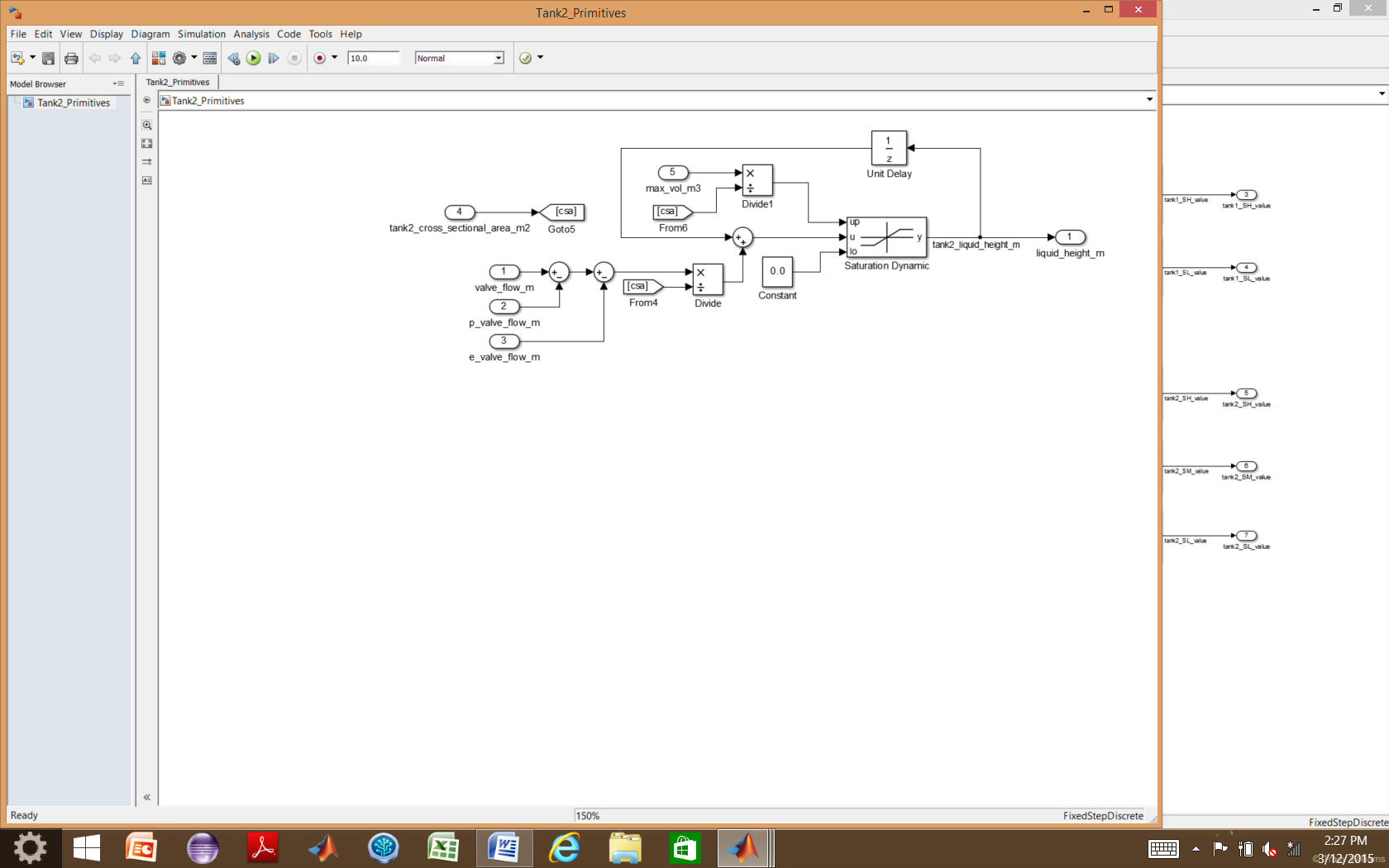


Figure 12: Tank 2 Model

## Sensors

The model used to simulate all five of the sensors used in the environment is shown in Figure 13.

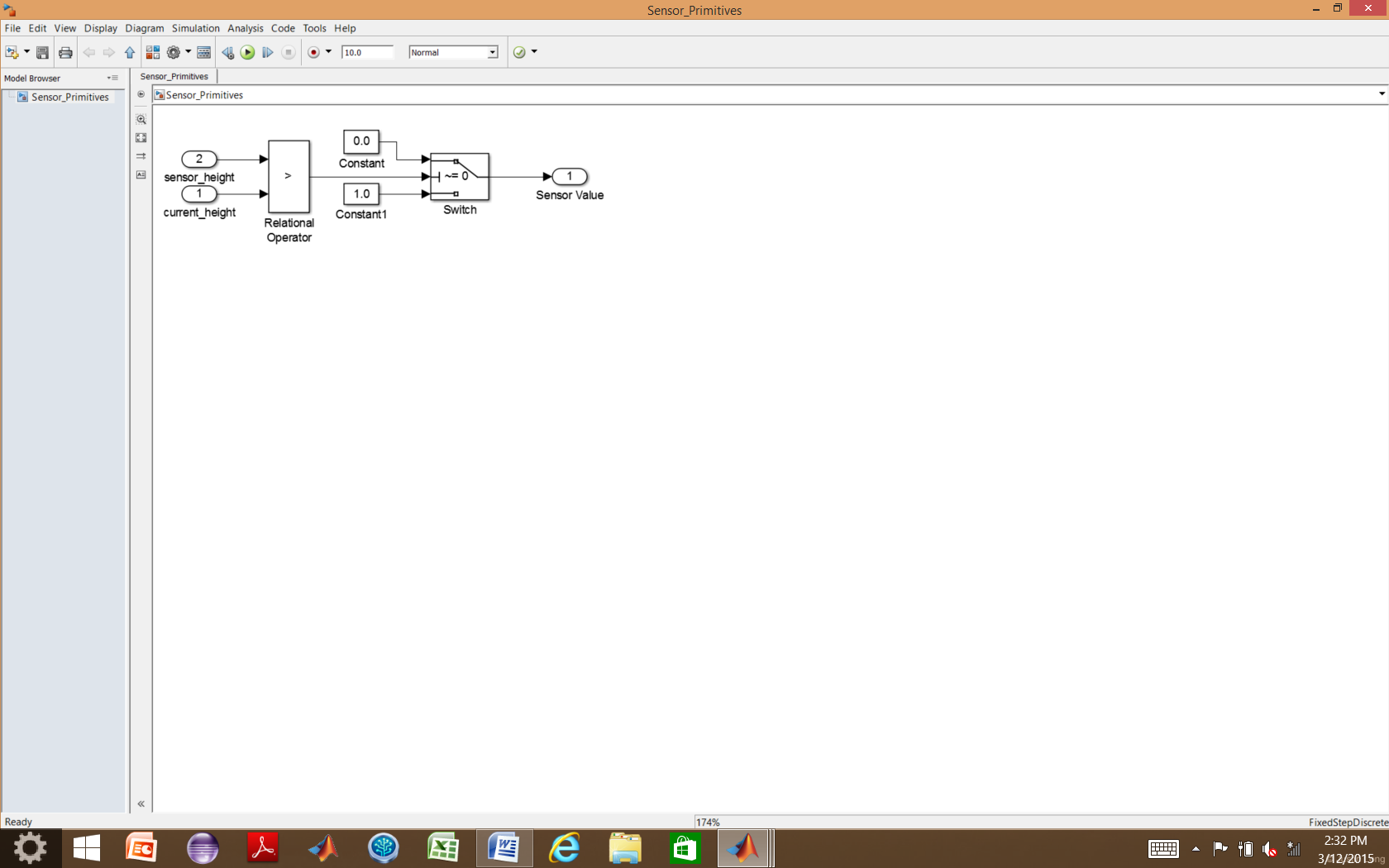


Figure 13: Sensor Model

Properties Files

A property file exists in Simulink as a MATLAB function block for each subsystem and is used to conduct verification of the subsystem. The properties files contain MATLAB functions, as seen in Figure 14.

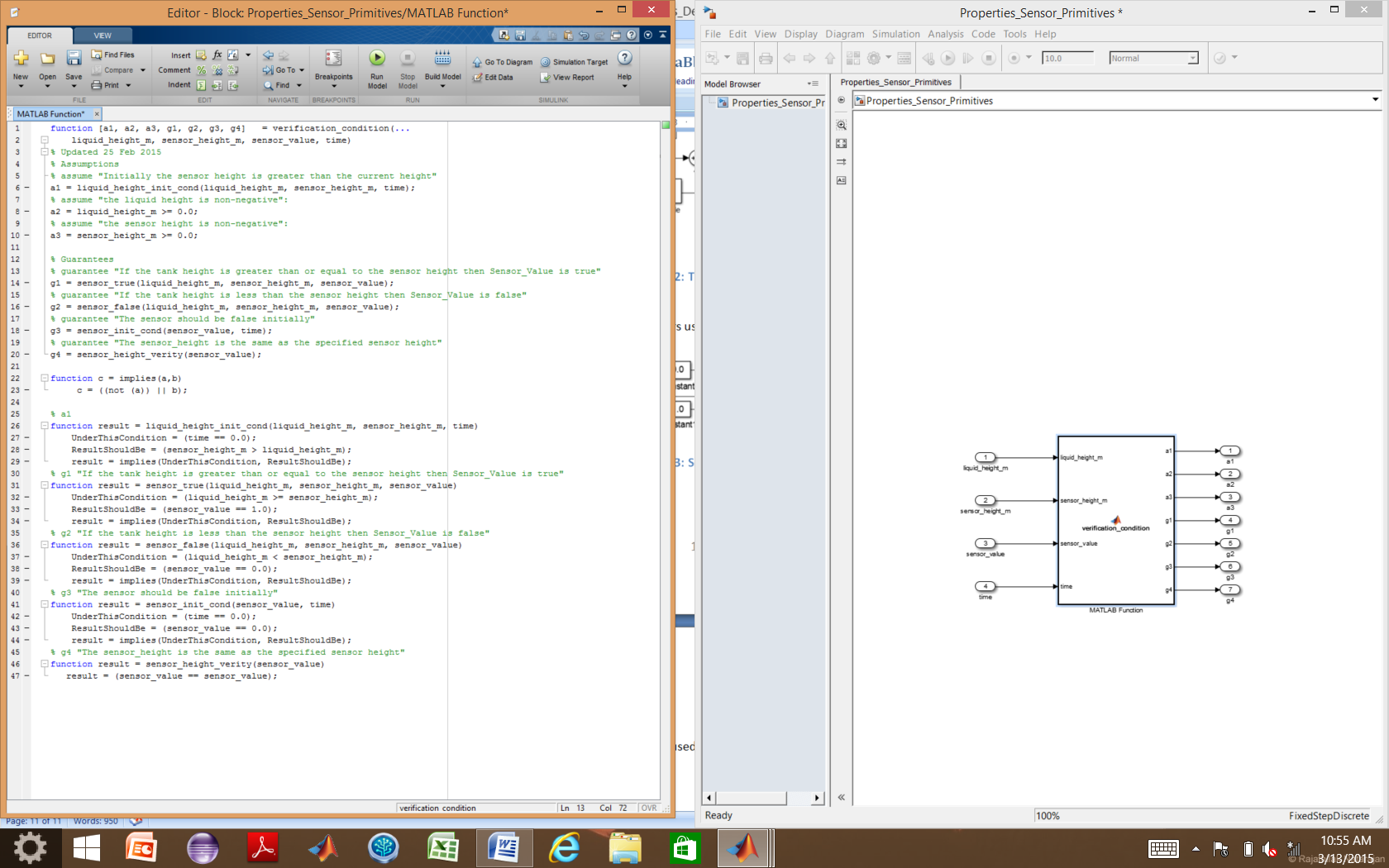


Figure 14: Example of Property File (from the Sensor Property File)

Each property file contains several functions. The main function accepts variables of interest, typically the inputs and outputs of the property in question, (in this example, the inputs to the main function for the sensor properties file are the inputs and outputs of the sensor model), and outputs the result of the analysis of the assumptions and properties for that subsystem. The other functions support analysis of the properties.

## Assumptions

If the design of the system is done in the VVAS envisioned approach of formal requirements specification and analysis, followed by formal architecture specification and analysis, followed by modeling and analysis, the assumptions specified in the property file should match the assumptions for the subsystem documented in the AGREE annex of the AADL architectural implementation in the previous design phase. The assumptions constrain the model checking and should be used with caution. In the example, assumptions are used to constrain the value of the liquid height and sensor height to non-negative numbers and to establish that the expected initial condition of the tank is that the liquid height will be below the sensor.

## Properties

If the architecture is developed in AADL with assumptions and guarantees written in the AGREE annex, the properties of the property file should match the guarantees of the AGREE file for that subsystem. For this reason, the properties are named “g1, g2, etc.” The properties describe performance characteristics that the model is expected to hold, and the objective of the analysis is to prove that the properties are satisfied by the model.

## Persistent Variables

When the previous value of a variable, a counter, or time tracking of a condition is needed in order to analyze a property of the system, persistent variables are used. Persistent variables hold their value in memory until the next time the function is called, but differ from global variables in that they are local to the function where they are defined. An example of the use of persistent variables in the Properties file for the Tank 1 Controller is displayed in Figure 15.

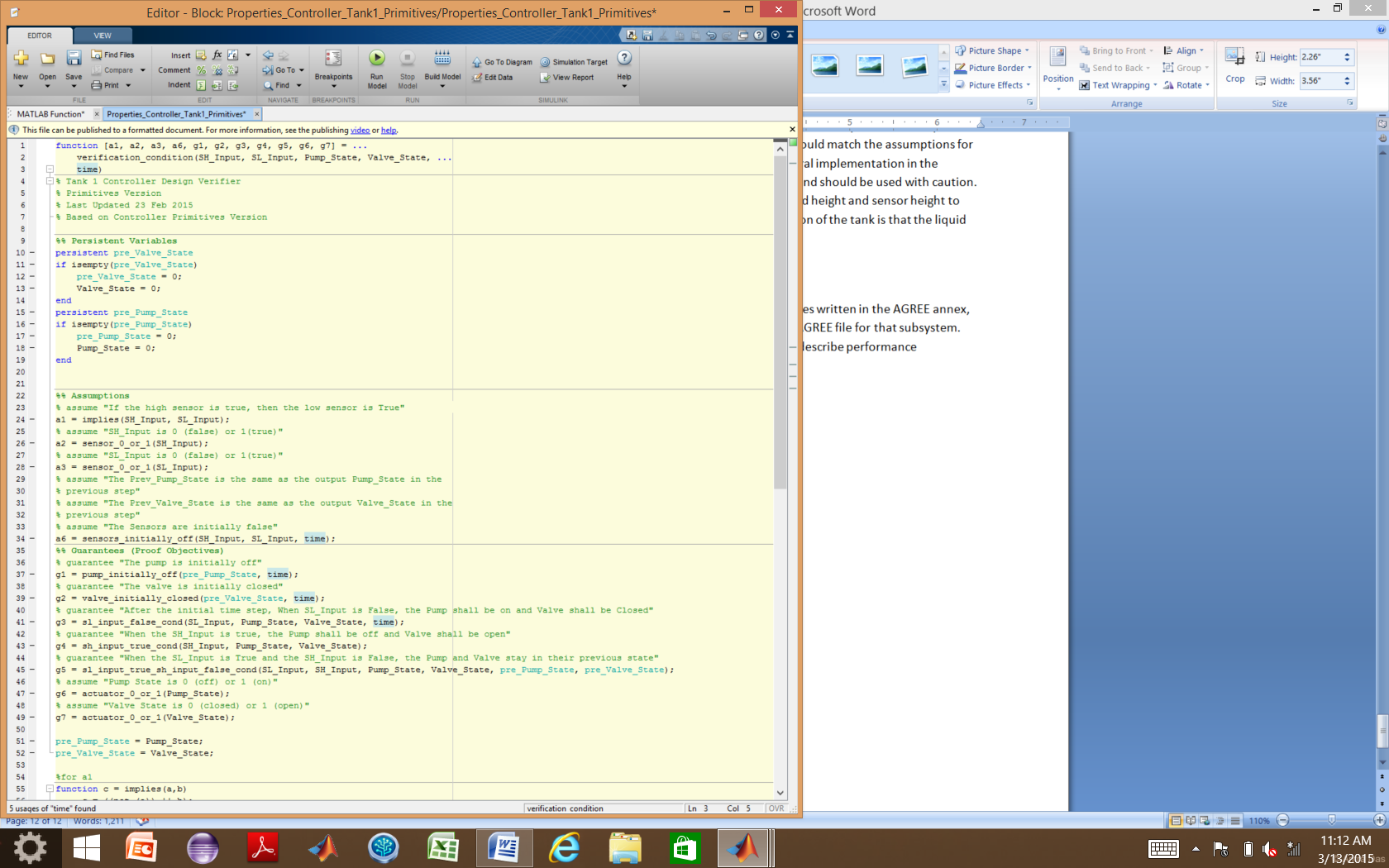
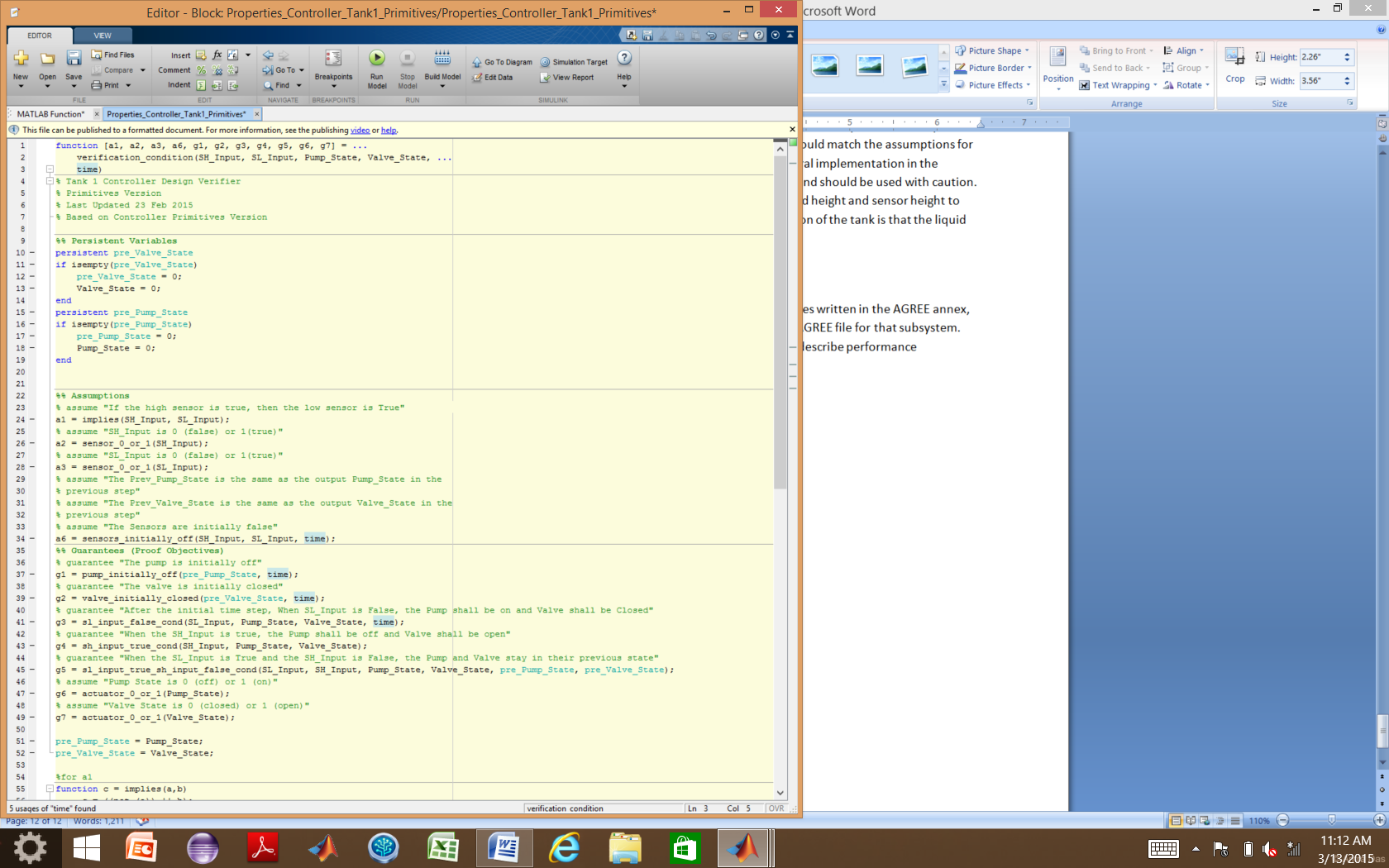


Figure 15: Persistent Variable Example (from the Tank 1 Controller Property File)

To use a persistent variable, it must first be declared, and then given an initial value using an *if statement* and the *isempty* function. The value of that variable should be updated to reflect the new state of the variable at the end of the function call.

# Verification Files

Verification requires a license for Simulink Design Verifier. With Simulink Design Verifier, the blocks used to do formal analysis of the subsystem model are the “Assumption” and “Proof Objective” blocks, shown in Figure 16.

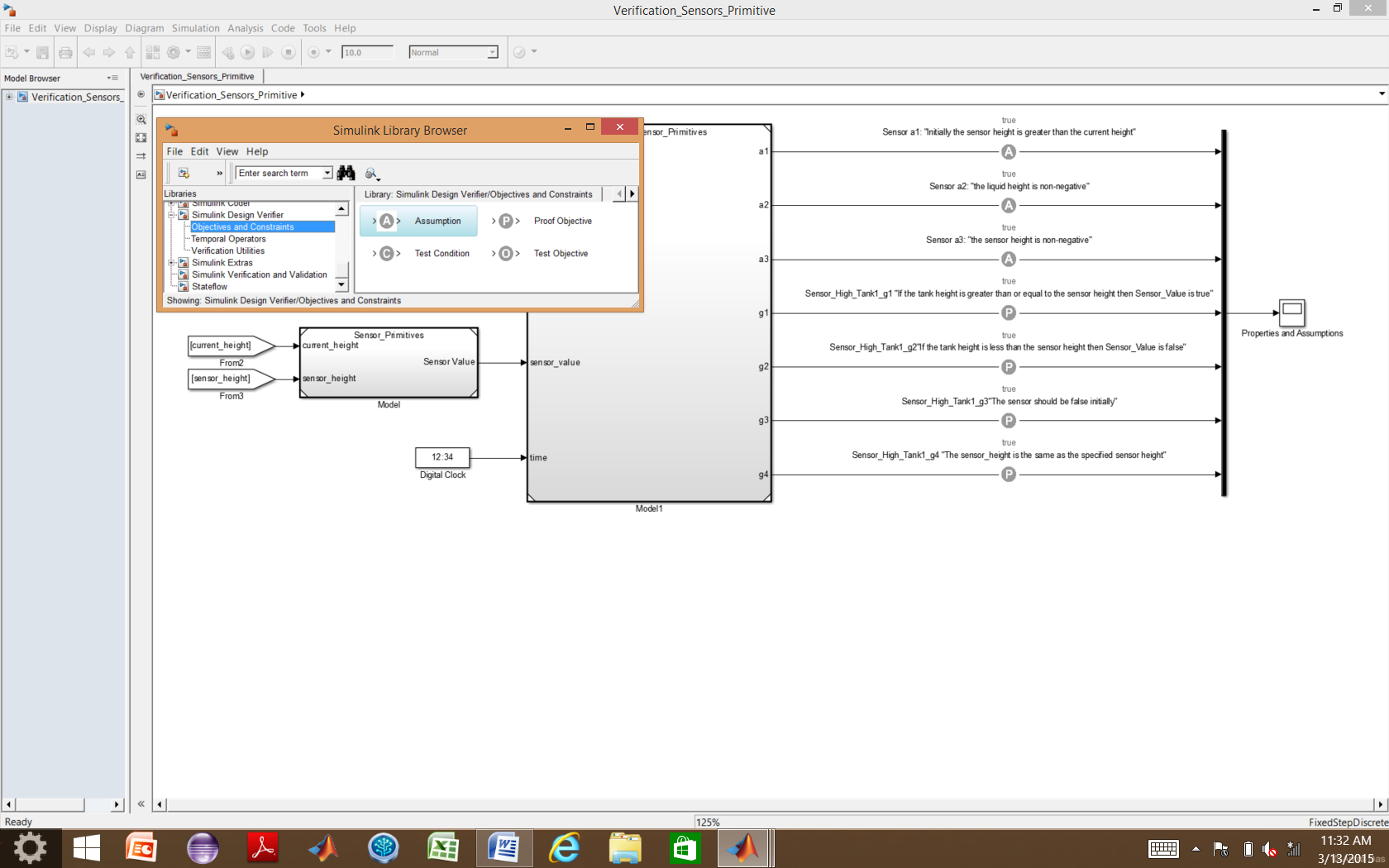


Figure 16: Location of Simulink Design Verifier Blocks in Library Browser

The verification files reference the model of the system and the properties model. The variables of interest (usually the inputs and outputs of the model block) are connected to the inputs of the properties block. The “Assumption” blocks are connected to the outputs of the property file corresponding to the assumptions “a1, a2, etc.” and the “Proof Objective” blocks are connected to the outputs of the property file corresponding to the properties “g1, g2, etc.” An example of a verification file for the sensor is shown in

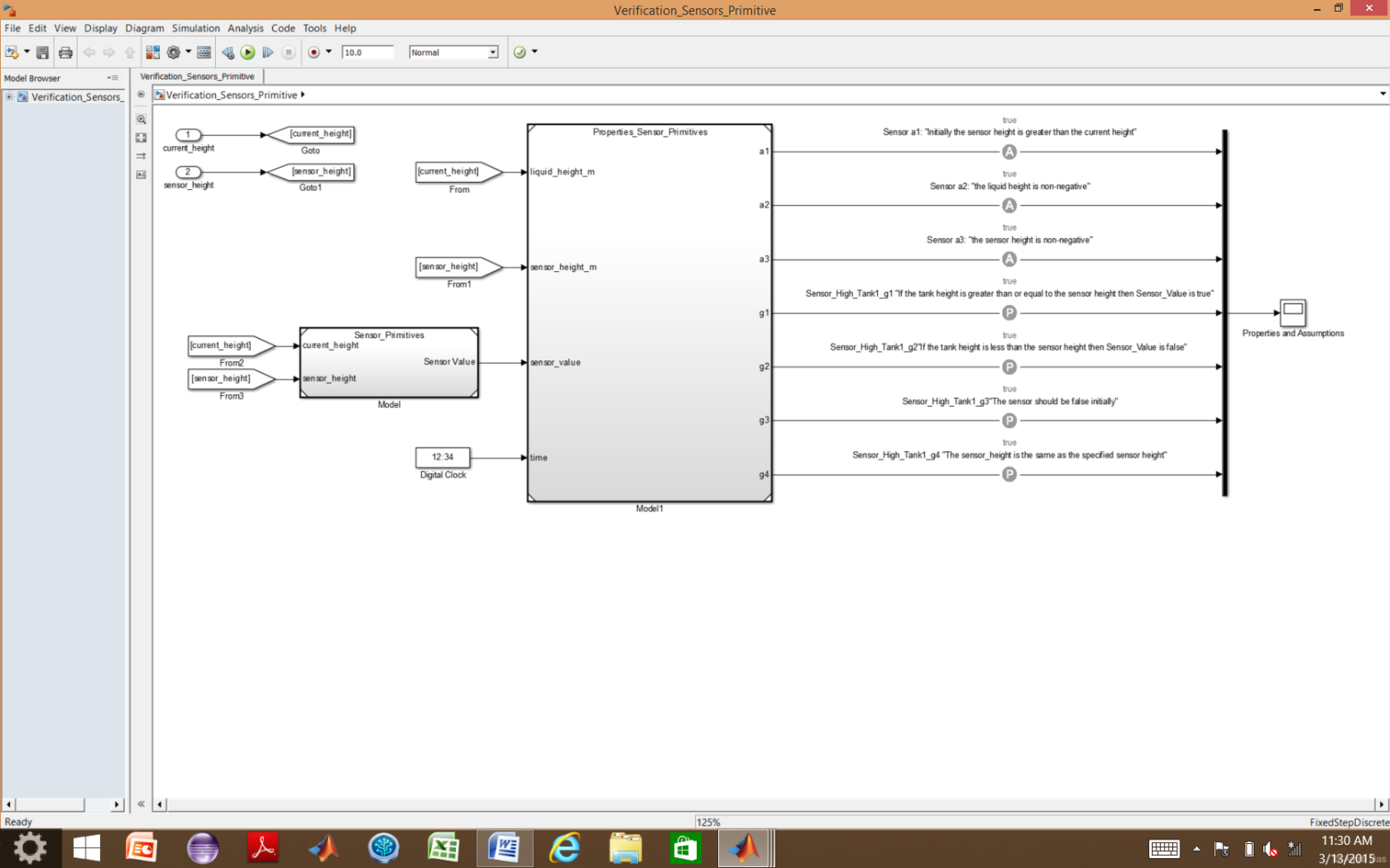


Figure 17: Verification File Example (from Sensor verification file)

# Properties and Assumptions used to Analyze the Subsystems

## Actuator

### Assumptions

* a1: "The actuator state is either 0 or 1"
* a2: "The time increment is positive"
* a3: "The actuator flow rate is positive"

### Properties/Guarantees (Proof Objectives)

* g1: "If the actuator is receiving an ON|OPEN signal then the flow through the actuator is the max flow rate"
* g2: "If the actuator is receiving an OFF|CLOSED signal then the flow rate through the actuator is 0.0"
* g3: "If there is flow through the actuator, then the actuator is receiving an ON|OPEN signal"
* g4: "If the flow is 0.0, then the pump is receiving on OFF|CLOSED signal"

## Controller for Tank 1

### Assumptions

* a1: "If the high sensor is true, then the low sensor is True"
* a2: "SH\_Input is 0 (false) or 1(true)"
* a3: "SL\_Input is 0 (false) or 1(true)"
* a4: "The Sensors are initially false"

### Properties/Guarantees (Proof Objectives)

* g1: "The pump is initially off"
* g2: "The valve is initially closed"
* g3: "After the initial time step, When SL\_Input is False, the Pump shall be on and Valve shall be Closed"
* g4: "When the SH\_Input is true, the Pump shall be off and Valve shall be open"
* g5: "When the SL\_Input is True and the SH\_Input is False, the Pump and Valve stay in their previous state"
* g6: "Pump State is 0 (off) or 1 (on)"
* g7: "Valve State is 0 (closed) or 1 (open)"

## Controller for Tank 2

### Assumptions

* a1: "If the high sensor is true, then the mid sensor is True" :
* a2: "If the mid sensor is true, then the low sensor is True"
* a3: "SH\_Input is 0 (false) or 1(true)"
* a4: "SM\_Input is 0 (false) or 1(true)"
* a5: "SL\_Input is 0 (false) or 1(true)"
* a8: "The sensors are all initially false (The tanks are initially empty)”

### Properties/Guarantees (Proof Objectives)

* g1: "The output emergency valve is initially closed"
* g2: "The output production valve is initially closed"
* g3: "When SL\_Input is False, the valves shall be Closed"
* g4: "When SM\_Input is False and the SL\_Input is true, the production valve shall be open and the emergency valve shall be closed"
* g5: "When the SH\_Input is true, both valves will be open"
* g6: "When the SM\_Input is True and the SH\_Input is False and the emergency valve was previously open, the valves shall be Open"
* g7: "When the SM\_Input is True and the SH\_Input is False and the emergency valve was previously closed, the production valve shall be open and the emergency valve shall be closed"
* g8: "Production Valve State is 0 (closed) or 1 (open)"
* g9: "Emergency Valve State is 0 (closed) or 1 (open)"

## Sensor

### Assumptions

* a1: "Initially the sensor height is greater than the current height"
* a2: "the liquid height is non-negative":
* a3: "the sensor height is non-negative":

### Properties/Guarantees (Proof Objectives)

* g1: "If the tank height is greater than or equal to the sensor height then Sensor\_Value is true"
* g2: "If the tank height is less than the sensor height then Sensor\_Value is false"
* g3: "The sensor should be false initially"
* g4: "The sensor\_height is the same as the specified sensor height"

## Tank 1

### Assumptions

* a1: "Pump flow rate is positive"
* a2: "Pump flow rate is initially 0"
* a3: "Tank cross section area is positive"
* a4: "Tank cross section area is constant"
* a5: "Valve flow rate is positive"
* a6: "Tank volume is positive"
* a7: "Tank volume is constant"
* a8: "Initial Height of the liquid is 0.0"
* a9: “valve\_flow\_rate is 0 or max flow rate”
* a10: “pump flow rate is 0 or max flow rate”

### Properties/Guarantees (Proof Objectives)

* g1: "The initial volume of tank 1 should be equivalent to the specified initial tank 1 volume"
* g2: "If the current tank 1 volume plus the tank 1 net flow is greater than the tank 1 maximum volume then tank 1 is full"
* g3: "If the current tank 1 volume plus the tank 1 net flow is less than the tank 1 minimum volume then tank 1 is empty"
* g4: "If the current tank 1 volume plus the tank 1 net flow is greater than empty and less than the tank 1 maximum volume then the tank 1 volume changes by the tank 1 net flow":
* g5: "Current height of tank 1 is equal or less than the max tank 1 height"
* g6: "Current height of tank 1 is not negative"

## Tank 2

### Assumptions

* a1: "Valve flow rate is positive"
* a2: "Tank cross section area is positive"
* a3 "Tank cross section area is constant"
* a4: "P valve flow rate is positive"
* a5: "E valve flow rate is positive"
* a6: "Tank volume is positive"
* a7: "Tank volume is constant"
* a8: "Initial Height of the liquid is 0.0"
* a9: "valve\_flow\_rate is 0 or max flow rate"
* a10: "production valve flow rate is 0 or max flow rate"
* a11: "emergency valve flow rate is 0 or max flow rate"

### Properties/Guarantees (Proof Objectives)

* g1: "The initial volume of tank 2 should be equivalent to the specified initial tank 2 volume"
* g2: "If the current tank 2 volume plus the tank 2 net flow is greater than the tank 2 maximum volume then tank 2 is full"
* g3: "If the current tank 2 volume plus the tank 2 net flow is less than the tank 2 minimum volume then tank 2 is empty"
* g4: "If the current tank 2 volume plus the tank 2 net flow is greater than empty and less than the tank 2 maximum volume then the tank 2 volume changes by the tank 2 net flow"
* g5: "Current height of tank 2 is equal or less than the max tank 2 height"
* g6: "Current height of tank 2 is not negative"

# Verification

Prior to conducting verification, the verification file should be run and no errors or warnings should appear. Once the verification model has been run, the results can be previewed using a scope block, however the scope may not provide a complete picture. To run the analysis, follow the menus in the model window as shown in Figure 18, finally selecting “Model.”

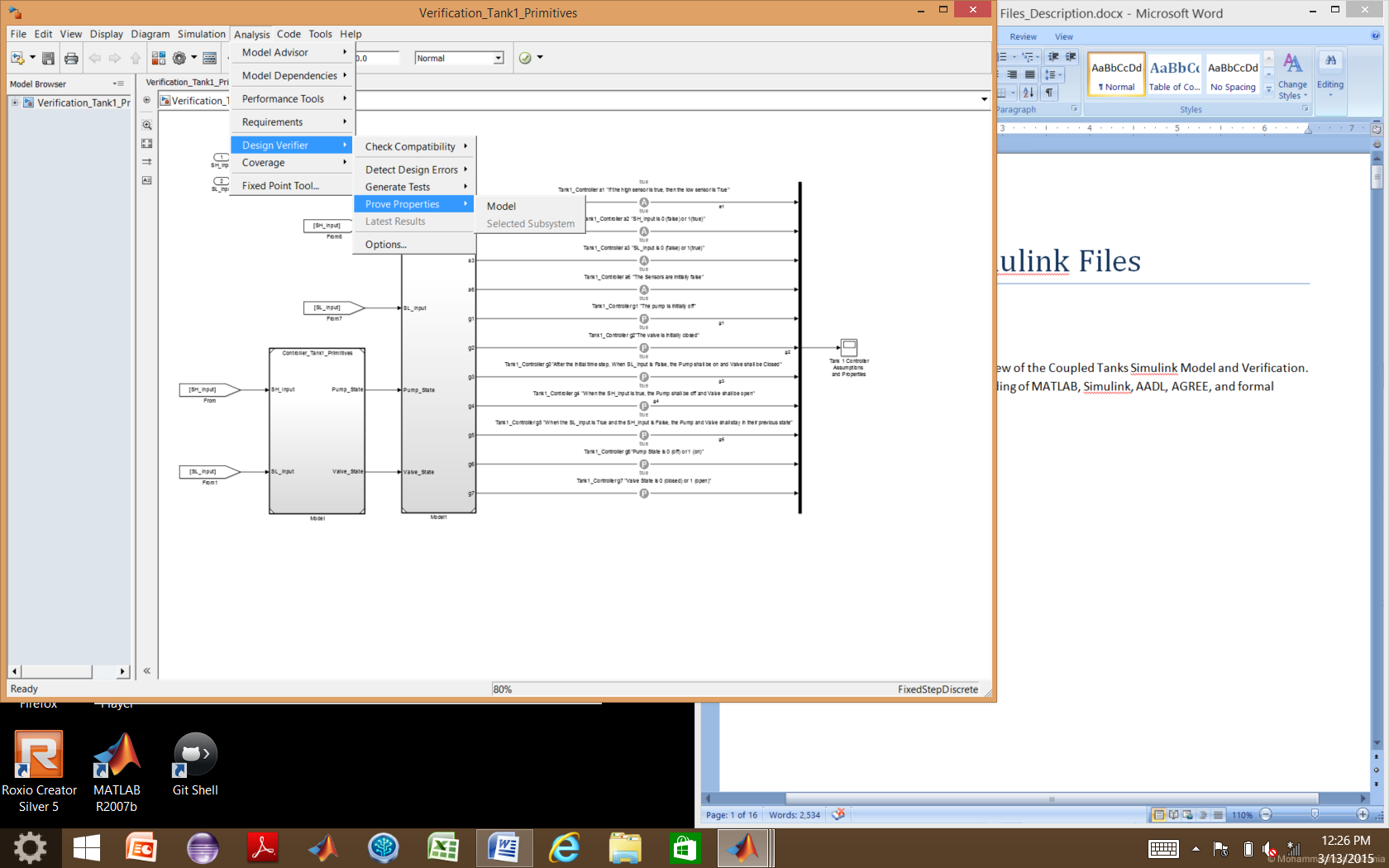


Figure 18: Design Verifier Menu Navigation for Analysis

Once you select “Model,” the Log Window shown in Figure 19 will appear, showing the progress of the analysis. In this example, all properties of Tank 1 are satisfied (proven). In the case that a property is falsified, Simulink Design Verifier will usually generate a counter example, which aids you in tracking down how the property is violated. Clicking on the “Generate detailed analysis report,” creates an HTML file which houses more information on the analysis such as counter examples for falsified properties.

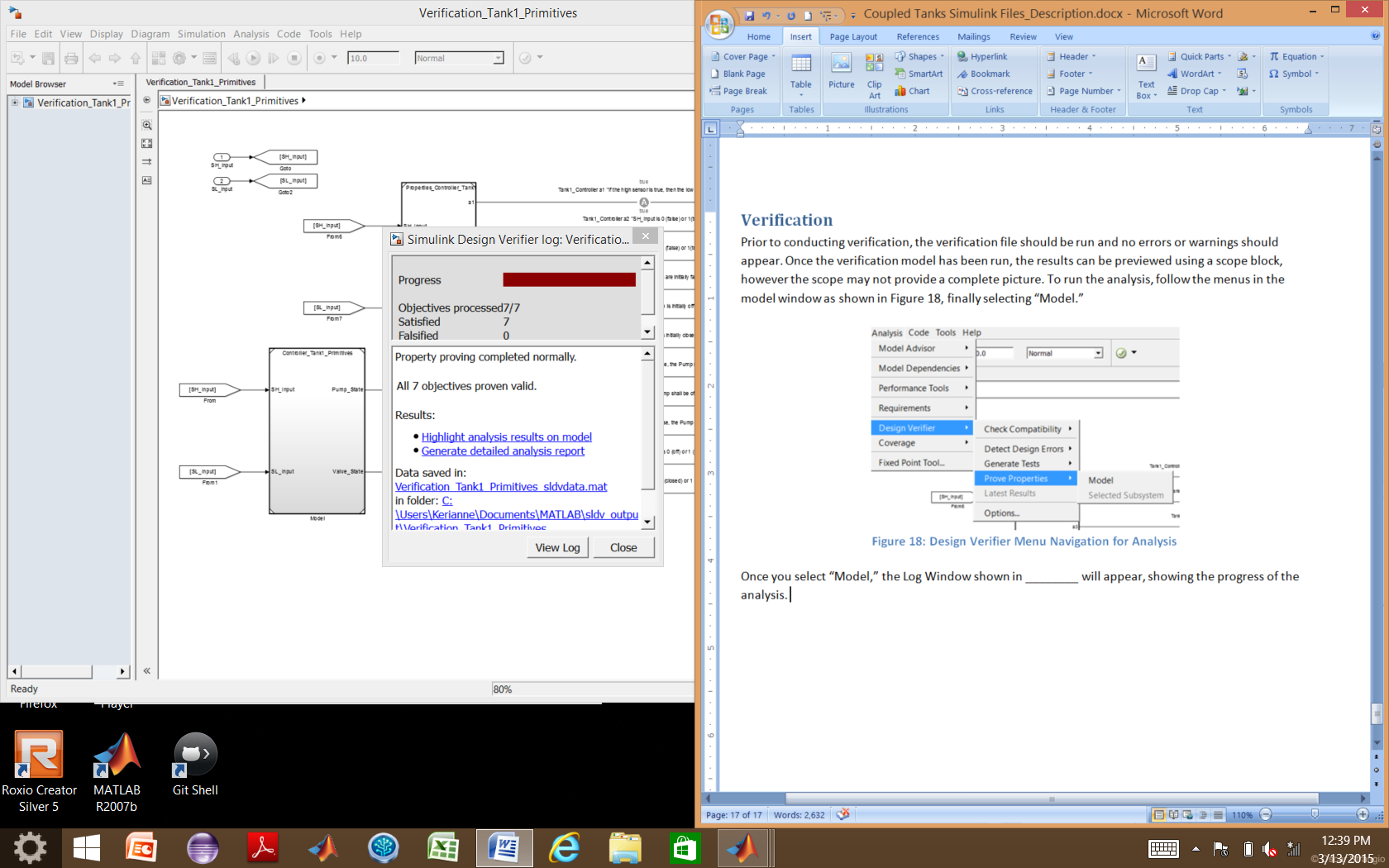


Figure 19: Simulink Design Verifier Log Window

1. Assurance Cases. (2014). Retrieved January 12, 2015, from http://www.sei.cmu.edu/dependability/tools/assurancecase/ [↑](#footnote-ref-1)