System Validation [IN4387]
Project Report

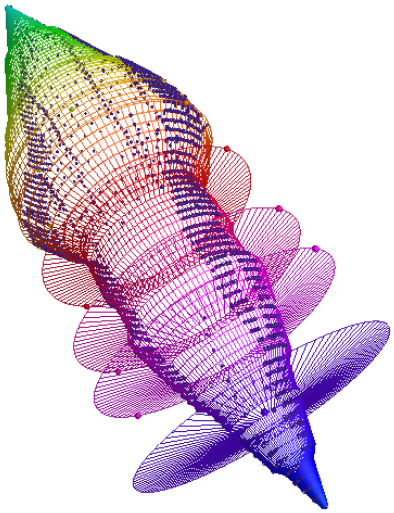
Group 1
Suryansh Sharma
S.sharma-13@student.tudelft.nl

Snehal Jauhri
S.jauhri@student.tudelft.nl

Suhail Nogd
S.T.S.nogd@student.tudelft.nl

Apoorva Arora
A.Arora-1@student.tudelft.nl

October 31, 2018



Contents

1 Introduction

This is a project done as part of TU Delft's IN4387 System Validation course. The project concerns designing, modelling and validating a controller for a Transfer system in an Industrial Silicon Wafer production plant.

The system consists of a UV Lamp that projects a design onto a wafer inside a vacuum chamber. The wafers are transferred to the Lamp via two Airlocks. The wafers are handled by robots from their initial position on the Input stacks to their final position on the Output stacks. The wafers move along the production line from their Initial state (on the Input Stacks), are printed on by the Lamp and reach the Final state (on the Output stacks).

Described here is the documentation for the modelling of the above system in mcrl2 and it's verification using Modal μ calculus. Section 2 describes the System and Functional Requirements. Section 3 is dedicated to the interactions between the various subsystems defined in section 2. The architecture of the resulting system is shown in section 4. In section 5 the requirements are translated into Modal μ-calculus. Section 6 describes the modelling process. In section 7 the model is verified with the translated requirements. The final conclusions are presented in section 8.

2 Requirements

2.1 System Components

The system consists of the following physical components:

- Lamp/Projector: L
- Inner Doors: DI1, DI2
- Outer Doors: DO1, DO2
- Input Stacks: I1, I2
- Output Stacks: O1, O2
- Airlocks: AL1, AL2
- Outer Robots: R1, R2
- Inner Robot: R3

2.2 System Requirements

The behaviour of the system can be understood by listing the following requirements:

1. Overall Requirement for our system: Once a wafer is picked up from an Input Stack, it eventually gets printed and placed on the Output Stack if the Output Stack is not full. (**As long as the wafer can move through the production process, it will.**) [**Liveness**]

2. Once a wafer is picked up from an Input Stack (I1 or I2), the next wafer from that Input Stack gets picked up only after the previous wafer has been printed and placed on the corresponding Output Stack. [**Safety**]

3. The Outer Robots (R1 and R2) should not place a wafer on the Input Stacks (I1 and I2). [**Safety**]

4. The Outer Robots (R1 and R2) should not pickup a wafer from an empty Input Stack. [**Safety**]

5. Once an **unprocessed** wafer is picked up from its Input Stack (I1 or I2) by the Outer Robots (R1 and R2), it will be placed on it's corresponding Airlock (A1 or A2). [**Liveness**]

6. The Outer Robots (R1 and R2) must not pickup an **unprocessed** wafer from the Airlocks (AL1 and AL2). [**Safety**]

7. The Outer and Inner Doors of the Airlocks must not be opened at the same time [**Safety**]

   (a) The Inner Door (DI1) must not be opened if the Outer Door(DO1) is open for Airlock (AL1).

   (b) The Inner Door (DI2) must not be opened if the Outer Door(DO2) is open for Airlock (AL2).

   (c) The Outer Door (DO1) must not be opened if the Inner Door(DI1) is open for Airlock (AL1).

   (d) The Outer Door (DO2) must not be opened if the Inner Door(DI2) is open for Airlock (AL2).

8. Once an **unprocessed** wafer is picked up from an Airlock by the Inner Robot (R3), the wafer will be placed on the Lamp. [**Liveness**]

9. The Inner Robot (R3) will only pickup a **finished** i.e. printed wafer from the Lamp (L). [**Safety**]

10. The Inner Robot (R3) will not place the **finished** wafer again on the Lamp (L). [**Safety**]

11. Once a **finished** wafer is picked up from the Lamp by the Inner Robot (R3), the wafer will be placed in its corresponding Airlock. [**Liveness**]

12. The Inner Robot (R3) should not pickup a **processed** wafer from the Airlocks. [**Safety**]

13. Once a **finished** wafer is picked up from an Airlock by the Outer Robots (R1 and R2), the wafer will be placed on it's corresponding Output Stack. [**Liveness**]

14. The Outer Robots (R1 and R2) should not pickup a wafer from the Output Stacks (O1 and O2). [**Safety**]

15. The Outer Robots (R1 and R2) should not place a wafer on the Output Stacks if the Output Stacks are Full. [**Safety**]

16. The Outer Robots (R1 and R2) should not place an **unprocessed** wafer on the Output Stacks (O1 and O2). [**Safety**]

17. **The system is deadlock free**. [**Safety**]

3 System Description and Interactions

3.1 System Description

We consider that our system consists of five components for control:

- **Input-Output Handlers** which are controllers that manage the (Outer) Robots R1 and R2 and their interactions with the pair of Stacks and the Airlocks. These IO Handlers monitor the Input and Output stacks, command the Robots and send/receive information to/from the Airlocks regarding presence of a wafer in the airlocks.
- **Airlock Chamber Controllers** which control the actuation of the Doors and the Airlocks' interactions with the rest of the system. These Airlock controllers receive and act on requests from the other components for opening/closing of doors. Moreover, they recieve information about wafers that have been placed in the Airlocks by the IO Handlers and Lamp Wafer Handler.
- A **Lamp Wafer Handler** which controls the (Inner)Robot R3 and its' interactions with the Lamp and the Airlocks. The Lamp Wafer Handler monitors the Lamp's printing process, commands the Inner Robot and sends/receives information to/from the Airlocks regarding presence of a wafer in the airlocks.

3.2 External Commands

The following are the commands given by the controller to the actuators of the system. The meaning can be interpreted as: **Command(Target)**

- Move(r, x) [r: R1, R2, R3, x: LocationID] : Move Robot (r) to mentioned Location.
- PickupWafer(r, x) [r: R1, R2, R3, x: LocationID] : Robot (r) Picks up the wafer from Location.
- PlaceWafer(r, x) [r: R1, R2, R3, x: LocationID] : Robot (r) Places the wafer at the Location.
- OpenDoor(x) [x: DI1, DI2, DO1, DO2] : Opens the corresponding door.
- CloseDoor(x) [x: DI1, DI2, DO1, DO2] : Closes the corresponding door.

The commands mentioned in Section 3.1 (above) are valid for the combinations of target Actuators and Destinations shown below:

	Lamp	Airlock1	Airlock2	Input1	Input2	Output1	Output2
Robot1		✓		✓		✓	
Robot2			✓		✓		✓
Robot3	✓	✓	✓				

The following commands are used to check sensor states:

- CheckIPStackState(x,s) [x: I1, I2 ; s: Empty, Full]
- CheckOPStackState(x,s) [x: O1, O2 ; s: Empty, Full]
- CheckLampState(s) [Incomplete, Complete]

3.3 Communications

The following are commands used by the controllers to communicate within the system. Their meaning can be interpreted as:

Command(ComponentID, State):

- commDoorState(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]
  - The Airlock Controllers **send** the state of the requested door (Open or Closed)
  - This is **received** by IO Handlers and Lamp Handlers

- commDoorRequest(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]
  - The IO Handlers and the Lamp Handler **send** the request to Open or Close a door
  - This is **received** by Airlock Controllers

- commWaferStatus(x,s) [x: AL1, AL2 ; s: Unprocessed, Finished]
  - The IO Handlers and the Lamp Handler **send** the Status of the wafer which has just been placed by them in the Airlocks(Unprocessed, Finished)

  - This is **received** by Airlock Controllers

- commWaferPresence(x,s) [x: AL1, AL2 ; s: Unprocessed, Finished , NoWafer]
  - The Airlock Controllers **send** a notification about the presence and type of Wafer present in the Airlock (Unprocessed, Finished, No wafer)
  - This is **received** by IO Handlers and the Lamp Handler

- commerror(x) [x: DO1,DO2,DI1,DI2 ; s: NA]
  - The Airlock Controllers **send** an error when the opening of the doors after receiveDoorRequest(x,Open) is unsuccessful.
  - This is **received** by IO Handlers and the Lamp Handler which asked to open the doors with sendDoorRequest(x,Open). **Note:** When an error is received by the Lamp Handler or the IO Handlers, the sendDoorRequest(x,open) action is initiated again.

4 Architecture

4.1 Global System Architecture

Figure 1 shows the Architecture of the system described above with five parallel controllers along with the various entities (Sensors and Actuators) they control.

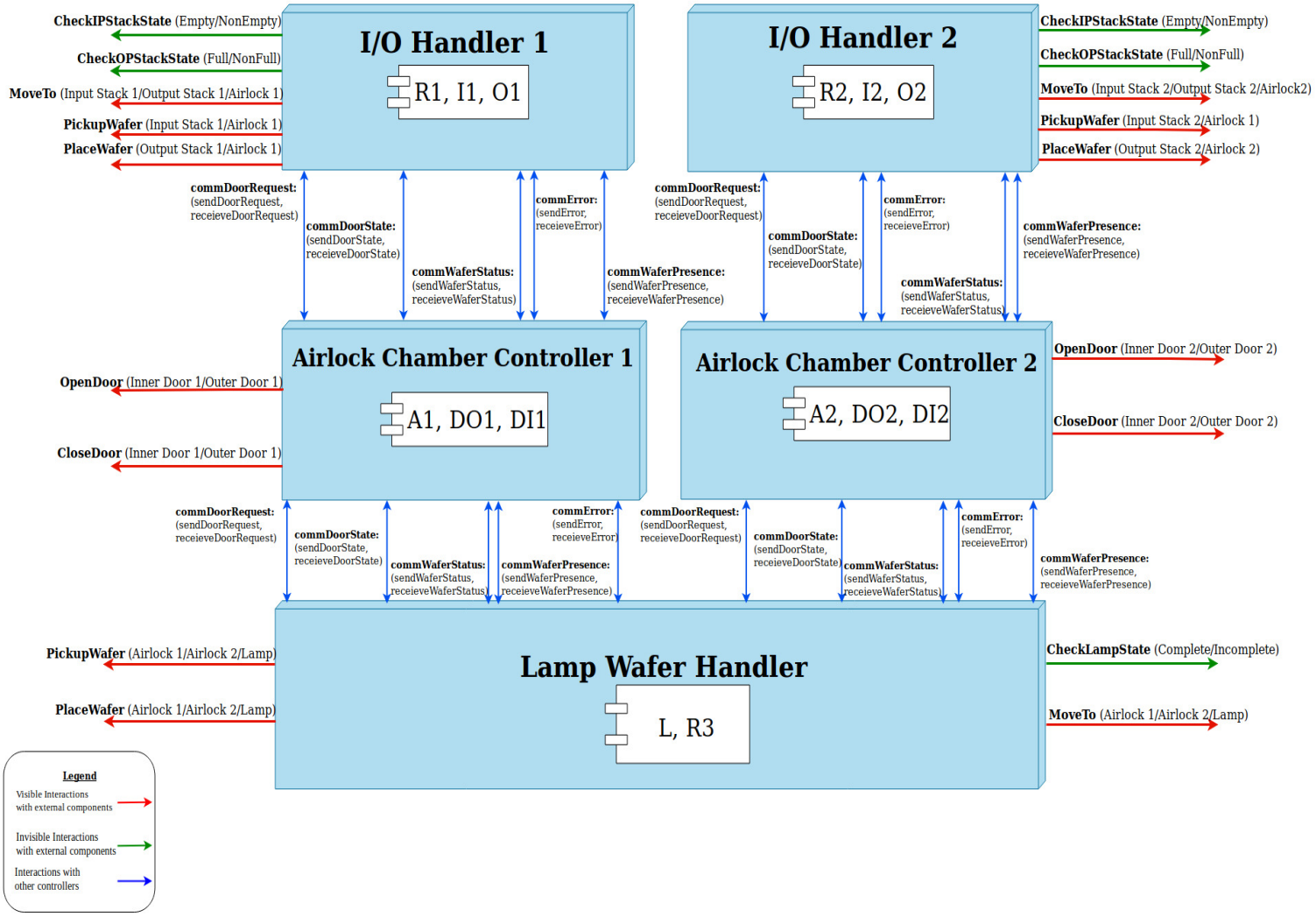


Figure 1: Architecture Diagram of System

5 Translated Requirements

5.1 Introduction

The requirements mentioned in Section 2 describe what is needed and expected out of the system. A successful verification of the system which conforms to the aforementioned specifications and requirements is to be carried out. This necessitates the use of Modal μ-Calculus.

5.2 Modal μ-Calculus

Here the corresponding Modal μ -Calculus are described:

1. Once a wafer is picked up from an Input Stack, it eventually gets printed and placed on the Output Stack if the Output Stack is not full. (**As long as the wafer can move through the production process, it will.**) [**Liveness**]
   This is the overall requirement of the system and will be satisfied when all the other liveness requirements of the system are satisfied. Thus, no modal formulas are specified specifically for this requirement.

2. Once a wafer is picked up from an Input Stack (I1 or I2), the next wafer from that Input Stack gets picked up only after the previous wafer has been printed and placed on the corresponding Output Stack. [**Safety**]

   Formula: $[true^*.PickupWafer(R1, I1)$
   $.(\overline{PlaceWafer(R1, O1)}^*.PickupWafer(R1, I1)]false$
   Status: Verified true.

   Formula: $[true^*.PickupWafer(R2, I2)$
   $.(\overline{PlaceWafer(R2, O2)}^*.PickupWafer(R2, I2)]false$
   Status: Verified true.

3. The Outer Robots (R1 and R2) should not place a wafer on the Input Stacks (I1 and I2). [**Safety**]

   Formula: $[true^*.PlaceWafer(R1, I1)]false$
   Status: Verified true.

   Formula: $[true^*.PlaceWafer(R2, I2)]false$
   Status: Verified true.

4. The Outer Robots (R1 and R2) should not pickup a wafer from an empty Input Stack. **[Safety]**

   Formula: $[true^*.CheckIPStackState(IP1, Empty).(\overline{CheckIPStackState(IP1, NonEmpty)})^*.PickupWafer(R1, I1)]false$
   Status: Verified true.

   Formula: $[true^*.CheckIPStackState(IP2, Empty).(\overline{CheckIPStackState(IP2, NonEmpty)})^*.PickupWafer(R2, I2)]false$
   Status: Verified true.

5. Once an **unprocessed** wafer is picked up from its Input Stack (I1 or I2) by the Outer Robots (R1 and R2), it will be placed on it's corresponding Airlock (A1 or A2). **[Liveness]**

   Formula: $[true^*.PickupWafer(R1, I1)] < true^*.PlaceWafer(R1, A1) > true$
   Status: Verified true.

   Formula: $[true^*.PickupWafer(R2, I2)] < true^*.PlaceWafer(R2, A2) > true$
   Status: Verified true.

6. The Outer Robots (R1 and R2) must not pickup an **unprocessed** wafer from the Airlocks (AL1 and AL2). **[Safety]**

   Formula: $[\overline{commWaferPresence(AL1, Finished)}^*.PickupWafer(R1, A1)]false$
   Status: Verified true.

   Formula: $[\overline{commWaferPresence(AL2, Finished)}^*.PickupWafer(R2, A2)]false$
   Status: Verified true.

7. The Outer and Inner Doors of the Airlocks must not be opened at the same time **[Safety]**

   (a) The Inner Door (DI1) must not be opened if the Outer Door(DO1) is open for Airlock (AL1).

   (b) The Inner Door (DI2) must not be opened if the Outer Door(DO2) is open for Airlock (AL2).

   (c) The Outer Door (DO1) must not be opened if the Inner Door(DI1) is open for Airlock (AL1).

(d) The Outer Door (DO2) must not be opened if the Inner Door(DI2) is open for Airlock (AL2).

Formula: $[true^*]\forall d : DoorID.[OpenDoor(d).(\overline{CloseDoor(d)}^*.OpenDoor(CorrespondingDoor(d))].false$
Status: Verified true.

8. Once an **unprocessed** wafer is picked up from an Airlock by the Inner Robot (R3), the wafer will be placed on the Lamp. [**Liveness**]

   Formula: $[true^*.PickupWafer(R3, A1)] < true^*.PlaceWafer(R3, Lamp) > true$
   Status: Verified true.

   Formula: $[true^*.PickupWafer(R3, A2)] < true^*.PlaceWafer(R3, Lamp) > true$
   Status: Verified true.

9. The Inner Robot (R3) will only pickup a **finished** i.e. printed wafer from the Lamp (L). [**Safety**]

   Formula: $[\overline{CheckLampState(Complete)}^*.PickupWafer(R3, Lamp)]false$
   Status: Verified true.

10. The Inner Robot (R3) will not place the **finished** wafer again on the Lamp (L). [**Safety**]

    Formula: $[true^*.PickupWafer(R3, Lamp).\overline{PlaceWafer(R3, A1) \vee PlaceWafer(R3, A2)}^*.PlaceWafer(R3, Lamp)]false$
    Status: Verified true.

11. Once a **finished** wafer is picked up from the Lamp by the Inner Robot (R3), the wafer will be placed in its corresponding Airlock. [**Liveness**]

    Formula: $[true*.PickupWafer(R3, A1).\overline{PickupWafer(R3, Lamp)}*.PickupWafer(R3, Lamp)] < true*.(PlaceWafer(R3, A1)) > true$
    Status: Verified true.

    Formula: $[true*.PickupWafer(R3, A2).\overline{PickupWafer(R3, Lamp)}*.PickupWafer(R3, Lamp)] < true*.(PlaceWafer(R3, A2)) > true$
    Status: Verified true.

12. The Inner Robot (R3) should not pickup a **processed** wafer from the Airlocks. [**Safety**]

    Formula: $[true^*.receiveWaferStatus(AL1, Finished).true^*.$
    $PickupWafer(R3, A1)]false$
    Status: Verified true.

13. Once a **finished** wafer is picked up from an Airlock by the Outer Robots (R1 and R2), the wafer will be placed on it's corresponding Output Stack. [**Liveness**] [**Liveness**]

    Formula: $[true*.PickupWafer(R1, A1)] < true*.$
    $PlaceWafer(R1, O1) > true$
    Status: Verified true.

    Formula: $[true*.PickupWafer(R2, A2)] < true*.$
    $PlaceWafer(R2, O2) > true$
    Status: Verified true.

14. The Outer Robots (R1 and R2) should not pickup a wafer from the Output Stacks (O1 and O2). [**Safety**]

    (a) **Formula:** $[true^*.PickupWafer(R1, O1)]false$
    Status: Verified true.

    (b) **Formula:** $[true^*.PickupWafer(R2, O2)]false$
    Status: Verified true.

15. The Outer Robots (R1 and R2) should not place a wafer on the Output Stacks if the Output Stacks are Full. [**Safety**]

    (a) **Formula:** $[true^*.CheckOPStackState(OP1, Full).$
    $\overline{CheckOPStackState(OP1, NonFull)}^*.PlaceWafer(R1, O1)]false$
    Status: Verified true.

    (b) **Formula:** $[true^*.CheckOPStackState(OP2, Full).$
    $\overline{CheckOPStackState(OP2, NonFull)}^*.PlaceWafer(R2, O2)]false$
    Status: Verified true.

16. The Outer Robots (R1 and R2) should not place an **unprocessed** wafer on the Output Stacks (O1 and O2). [**Safety**]

    (a) **Formula:** $[true^*.PickupWafer(R1, I1).\overline{PickupWafer(R1, A1)}^*.$
    $PlaceWafer(R1, O1)]false$
    Status: Verified true.

(b) **Formula:** $[true^*.PickupWafer(R2, I2).\overline{PickupWafer(R2, A2)}^*.PlaceWafer(R1, O1)]false$
Status: Verified true.

17. **The system is deadlock free**. **[Safety]**

Formula: $[true^*]. < true > .true$
Status: Verified true.

6 Modelling the System

6.1 Component Description

Our Model consists of the following Components:

- 2 IO Handlers which control the (Outer)Robots R1 and R2 and their interactions with the pair of Stacks and the Airlocks.
- 2 Airlock Controllers which control the actuation of the Doors and the Airlocks' interactions with the rest of the system.
- 1 Lamp Handler which controls the (Inner)Robot R3 and its' interactions with the Lamp and the Airlocks.

Currently the two controllers called IO Handler 1 and IO Handler 2 are working in parallel with 2 separate Airlock Controllers for each Airlock and with the Lamp Handler. This results in a system with five parallel components working to move the wafer along the production process.

6.2 Initial State

The system in its initial state has the following configuration:

- The Output Stacks start with having no wafers present(empty). The Input Stacks are assumed to have wafers present (non-empty).
- The Airlocks don't have any wafer already present in them. All of the Doors of both the Airlocks are closed.
- The Lamp does not have any wafer present on it.

The Input The system currently has 89 levels, 2079 states and 4740 transitions.

6.3 Restrictions and Extensions

The system restricts the controllers to only control and interact with one half of the symmetrical system. This implies that the IO Handlers and Airlock Controllers only interact with a single robot (R1 or R2), single pair of Input and Output stack (I1,O1 or I2,O2), Airlocks (AL1 or AL2) and the corresponding pair of doors (DO1,DI1 or DO2,DI2). This simplifies the system to a great extent and helps in it's modelling.

There is a caveat though. This simplification will reduce the throughput of the system. This happens when one of the Stacks no longer have wafers present but the other pair of stacks still do.

Hence, as an extension, we have considered the possibility of the stacks being replenished. This ensures that the Robots continuously check the Input and Output stack till they are in a state where the system can proceed. Instead of dealing with the number of wafers present in the stacks, we have abstracted that away to a binary state of the stacks, that is, are usable or not. Usable here indicates different meanings for Input and Output stacks. For the Input stack we check if the stacks are Empty or Non-Empty. For the output stacks the check is performed to determine if they are Full or Not-Full. This ensures that if at any point of time the stacks are replenished (input) or cleared (output) the system will continue to work.

Thus, We have considered the case when the output stacks are emptied once they are full and the case when the input stacks are replenished. Our system loops around the check for the stacks to return to a workable state (Input not empty and Output not full)

Another extension we came up with was a condition where the door failed to open after a request from the IO Handler or the Lamp Wafer Handler. We added an error communication from the Airlock Controller to the requester in this case and modeled our system to retry opening the door in such a case. This guards against a deadlock due to cases such as the door being jammed while opening etc.

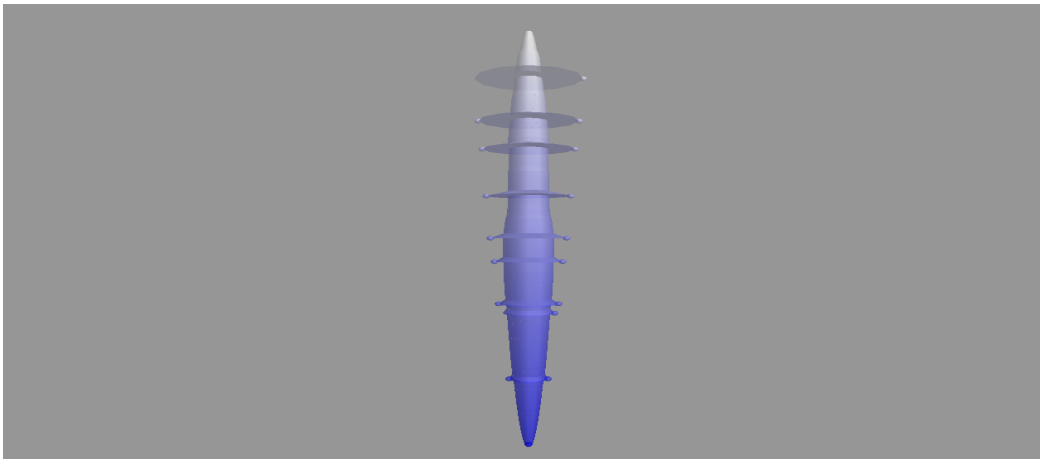


Figure 2: LTSView of the System

7 Verification

The Verification process involved three main steps. We created a simplified version of the model where only a single IO Handler, Airlock Controller and Lamp Handler was present. We disregarded the second pair of Input Output Stacks along with the Robot and Airlock. This way we could visualize the system using LTSGraph. This worked well because we needed to identify the cause for the deadlock occurring in our system. We used LPSxSim simulation to logically verify the correct sequence of actions for the process (only for the simplified system). Once we corrected the mistake and cleared the deadlocking condition, we used the complete, more complex system for all of our verification.

The Complete Model was visually inspected using LTSView to mark for deadlocks.These are presented in the Visualization subsection. The μ Calculus derived from the translated requirements from Section 5 are presented in Appendix B

7.1 Tools

In order to replicate our results, we provide the exact version of the tools we used along with the system configuration we used them on.

The following version of mcrl2 was used:

- mcrl2 –version: 201808.0

The following tools were used for modelling and verification: mcrl2xi, mcrl22lps, lpsxsim, lps2lts, ltsgraph, ltsview, lts2pbes, pbessolve.
The following is the specification of the system used for the above tools:

- Windows 10, Intel i-core i7, 2.8GHz processor with 16GB RAM

7.2 Verification Checks

The method used for formal verification of the system requirements presented in Section 2 is converting the model to PBES and then verifying each Modal μ Formula individually. The formulae for Modal μ Calculus presented in Section 5 have been verified individually and each of the formula holds true for the model presented.

7.3 Visualizations

The visual representations provide an additional guide to verify that the system is, indeed, deadlock free. Figure 3 shows the system when visualized with LTSView tool. The figure represents the states and transitions present in the model. Figure 4 shows the model when marked for deadlocks. The absence of red dots confirms that the system is deadlock free.

Figure 3: LTSView

Figure 4: Marking Deadlocks

8 Conclusion and Future Work

As part of the given assignment, we have modelled a Wafer production system as described in the assignment prompt. This project consisted of multiple phases.

In the first phase the system components were determined and requirements were formalized. In the second phase the interactions between the systems and the communications between different controllers were defined. Based on the system components along with the formulated requirements and interactions an architecture was developed. The third phase involved modelling with the mcrl2 tool. In the fourth phase requirements were translated into μ-calculus Calculus. In the fifth phase the model was verified by translating the μ-calculus Calculus from phase 4 into Modal μ-Code.

We feel there is a good scope to extend the system to increase the throughput and decrease the wafer transfer time. This can be done by allowing each of the robots(R1, R2) to access each of the stacks and airlocks. This would eliminate (reduce) the waiting time of the outer robots at the airlocks for a finished wafer.But this would mean increased complexity of the system. To speed up the system further, the robots can be allowed to handle two wafers at a time. This can be done by modelling the arms/hands of the robots. These extensions to the system were thought of during the project and can be the future scope of the project.

A Code: mcrl2

sort LocationID = **struct** A1 | A2 | I1 | I2 | O1 | O2 | Lamp;
IPStackID = struct IP1 | IP2;
OPStackID = struct OP1 | OP2;
AirlockID = **struct** AL1 | AL2 | None;
RobotID = struct R1 | R2 | R3;
OperationType = **struct** Get | Put;
IPStackState = **struct** Empty | NonEmpty;
OPStackState = **struct** Full | NonFull;
DoorID = **struct** DI1 | DI2 | DO1 | DO2;
DoorState = **struct** Open | Closed;
LampState = **struct** Incomplete | Complete;
CycleType = **struct** Input | Output;
WaferType = **struct** Unprocessed | Finished | NoWafer;

map CorrespondingDoor : DoorID − > DoorID;
CorrespondingOPDestination : IPStackID − >
DestinationID;
MapOPDestination : OPStackID − > DestinationID;
MapIPDestination : IPStackID − > DestinationID;
MapAirlock : AirlockID − > DestinationID;

eqn CorrespondingDoor(DO1) = DI1;
CorrespondingDoor(DI1) = DO1;
CorrespondingDoor(DO2) = DI2;
CorrespondingDoor(DI2) = DO2;

CorrespondingOPDestination(IP1) = O1;
CorrespondingOPDestination(IP2) = O2;

MapOPDestination(OP1) = O1;
MapOPDestination(OP2) = O2;
MapIPDestination(IP1) = I1;
MapIPDestination(IP2) = I2;

MapAirlock(AL1) = A1;
MapAirlock(AL2) = A2;
MapAirlock(None) = Null;

act Move: RobotID # LocationID;

PickupWafer : RobotID # LocationID;
PlaceWafer : RobotID # LocationID;

OpenDoor : DoorID;
CloseDoor : DoorID;

CheckIPStackState : StackID # IPStackState;
CheckOPStackState : StackID # OPStackState;
CheckLampState : LampState;

receiveDoorState : DoorID # DoorState;
sendDoorState : DoorID # DoorState;
commDoorState : DoorID # DoorState;

receiveDoorRequest : DoorID # DoorState;
sendDoorRequest : DoorID # DoorState;
commDoorRequest : DoorID # DoorState;

receiveWaferStatus : AirlockID # WaferType;
sendWaferStatus : AirlockID # WaferType;
commWaferStatus : AirlockID # WaferType;

receiveWaferPresence : AirlockID # WaferType;
sendWaferPresence : AirlockID # WaferType;
commWaferPresence : AirlockID # WaferType;

proc IOHandler1(Operation : OperationType, Cycle : CycleType) =

((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP1,Empty)
.IOHandler1(Operation = Get)

+ ((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP1,NonEmpty)
.Move(R1,I1).PickupWafer(R1,I1).IOHandler1(Operation = Put)

+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO1,Closed)
.sendDoorRequest(DO1,Open).IOHandler1(Operation = Put)

+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO1,Open)
.Move(R1,A1).PlaceWafer(R1,A1).sendWaferStatus(AL1,New).IOHandler1(Cycle
= Output, Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPresence(AL1,NoWafer).IOHandler1(Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPresence(AL1,Finished).receiveDoorState(DO1,Closed).sendDoorRequest(DO1,Open)
.receiveDoorState(DO1,Open).Move(R1,A1).PickupWafer(R1,A1).IOHandler1(Operation
= Put)

+ ((Cycle == Input) && (Operation == Put)) -> receiveError(DO1)
.sendDoorRequest(DO1,Open).IOHandler1(Operation = Put)
+ ((Cycle == Output) && (Operation == Get)) -> receiveError(DO1)
.sendDoorRequest(DO1,Open).IOHandler1(Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPresence(AL2,NoWafer).IOHandler1(Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> (receiveWaferPresence(AL1,Finished).((receiveDoorState(DO1,Closed).sendDoorRequest(DO1,Open)
.IOHandler1(Operation = Get)) + (receiveDoorState(DO1,Open).Move(R1,A1)
.PickupWafer(R1,A1).IOHandler1(Operation = Put))))

+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,Full)
.IOHandler1(Operation = Put)
+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,NonFull)
.Move(R1,O1).PlaceWafer(R1,O1).IOHandler1(Cycle = Input, Operation =
Get);

IOHandler2(Operation : OperationType, Cycle : CycleType) =

((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP2,Empty)
.IOHandler2(Operation = Get)
+ ((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP2,NonEmpty)
.Move(R2,I2).PickupWafer(R2,I2).IOHandler2(Operation = Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO2,Closed)
.sendDoorRequest(DO2,Open).IOHandler2(Operation = Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO2,Open)
.Move(R2,A2).PlaceWafer(R2,A2).sendWaferStatus(AL2,New).IOHandler2(Cycle
= Output, Operation = Get)

+ ((Cycle == Input) && (Operation == Put)) -> receiveError(DO2)
.sendDoorRequest(DO2,Open).IOHandler2(Operation = Put)
+ ((Cycle == Output) && (Operation == Get)) -> receiveError(DO2)
.sendDoorRequest(DO2,Open).IOHandler2(Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPres-
ence(AL2,NoWafer).IOHandler2(Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> (receiveWaferPres-
ence(AL2,Finished).((receiveDoorState(DO2,Closed).sendDoorRequest(DO2,Open)
.IOHandler2(Operation = Get)) + (receiveDoorState(DO2,Open).Move(R2,A2)
.PickupWafer(R2,A2).IOHandler2(Operation = Put))))

+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP2,Full)
.IOHandler2(Operation = Put)
+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP2,NonFull)
.Move(R2,O2).PlaceWafer(R2,O2).IOHandler2(Cycle = Input, Operation =
Get);

AirlockChamber1Controller(WaferPresence : WaferType, OuterDoorState :
DoorState, InnerDoorState : DoorState) =

((OuterDoorState == Closed) && (InnerDoorState == Open)) -> receive-
DoorRequest(DO1,Open)
.AirlockChamber1Controller(InnerDoorState = Open)

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> re-
ceiveDoorRequest(DO1,Open)
.OpenDoor(DO1).AirlockChamber1Controller(OuterDoorState = Open)
+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> re-
ceiveDoorRequest(DI1,Open)
.AirlockChamber1Controller(OuterDoorState = Open)
+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> re-
ceiveDoorRequest(DI1,Open)
.OpenDoor(DI1).AirlockChamber1Controller(InnerDoorState = Open)

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> re-
ceiveDoorRequest(DO1,Open).sendError(DO1).AirlockChamber1Controller()

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> re-
ceiveDoorRequest(DI1,Open).sendError(DI1).AirlockChamber1Controller()

+ ((OuterDoorState == Closed) && (InnerDoorState == Open)) -> receiveDoorRequest(DO1,Open).sendError(DO1).AirlockChamber1Controller()

+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> receiveDoorRequest(DI1,Open).sendError(DI2).AirlockChamber1Controller()

+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> receiveWaferStatus(AL1,New).CloseDoor(DO1)
.AirlockChamber1Controller(WaferPresence = New, OuterDoorState = Closed)
+ ((OuterDoorState == Closed) && (InnerDoorState == Open)) -> receiveWaferStatus(AL1,Finished).CloseDoor(DI1)
.AirlockChamber1Controller(WaferPresence = Finished, InnerDoorState = Closed)

+ sendDoorState(DO1,OuterDoorState).AirlockChamber1Controller()
+ sendDoorState(DI1,InnerDoorState).AirlockChamber1Controller()
+ sendWaferPresence(AL1,WaferPresence).AirlockChamber1Controller();

AirlockChamber2Controller(WaferPresence : WaferType, OuterDoorState : DoorState, InnerDoorState : DoorState) =

((OuterDoorState == Closed) && (InnerDoorState == Open)) ->
receiveDoorRequest(DO2,Open).AirlockChamber2Controller(InnerDoorState = Open)
+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) ->
receiveDoorRequest(DO2,Open).OpenDoor(DO2)
.AirlockChamber2Controller(OuterDoorState = Open)
+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) ->
receiveDoorRequest(DI2,Open)
.AirlockChamber2Controller(OuterDoorState = Open)
+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) ->
receiveDoorRequest(DI2,Open).OpenDoor(DI2)
.AirlockChamber2Controller(InnerDoorState = Open)

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> receiveDoorRequest(DO2,Open).sendError(DO2).AirlockChamber2Controller()

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> receiveDoorRequest(DI2,Open).sendError(DI2).AirlockChamber2Controller()

+ ((OuterDoorState == Closed) && (InnerDoorState == Open)) -> receiveDoorRequest(DO2,Open).sendError(DO2).AirlockChamber2Controller()

+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> receiveDoorRequest(DI2,Open).sendError(DI2).AirlockChamber2Controller()

+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> receiveWaferStatus(AL2,New).CloseDoor(DO2)
.AirlockChamber2Controller(WaferPresence = New, OuterDoorState = Closed)
+ ((OuterDoorState == Closed) && (InnerDoorState == Open)) -> receiveWaferStatus(AL2,Finished).CloseDoor(DI2)
.AirlockChamber2Controller(WaferPresence = Finished, InnerDoorState = Closed)

+ sendDoorState(DO2,OuterDoorState).AirlockChamber2Controller()
+ sendDoorState(DI2,InnerDoorState).AirlockChamber2Controller()
+ sendWaferPresence(AL2,WaferPresence).AirlockChamber2Controller();

LampWaferHandler(Cycle : CycleType, CurrentAirlock : AirlockID) =

((Cycle == Input) && (CurrentAirlock == None)) -> receiveWaferPresence(AL1,New).LampWaferHandler(CurrentAirlock = AL1)
+ ((Cycle == Input) && (CurrentAirlock == AL1)) -> receiveDoorState(DI1,Closed)
.sendDoorRequest(DI1,Open).LampWaferHandler(CurrentAirlock = AL1)
+ ((Cycle == Input) && (CurrentAirlock == AL1)) -> receiveDoorState(DI1,Open)
.Move(R3,A1).PickupWafer(R3,A1).Move(R3,Lamp).PlaceWafer(R3,Lamp)
.LampWaferHandler(Cycle = Output)

+ ((Cycle == Input) && (CurrentAirlock == AL1)) -> receiveError(DI1)
.sendDoorRequest(DI1,Open).LampWaferHandler(CurrentAirlock = AL1)

+ ((Cycle == Input) && (CurrentAirlock == AL2)) -> receiveError(DI2)
.sendDoorRequest(DI2,Open).LampWaferHandler(CurrentAirlock = AL2)

+ ((Cycle == Output) && (CurrentAirlock == AL1)) -> CheckLampState(Incomplete).LampWaferHandler(Cycle = Output)

+ ((Cycle == Output) && (CurrentAirlock == AL1)) -> CheckLampState(Complete).Move(R3,Lamp).PickupWafer(R3,Lamp)
.Move(R3,A1).PlaceWafer(R3,A1).sendWaferStatus(AL1,Finished)
.LampWaferHandler(Cycle = Input, CurrentAirlock = None)

+ ((Cycle == Input) && (CurrentAirlock == None)) -> receiveWaferP-
resence(AL2,New).LampWaferHandler(CurrentAirlock = AL2)

+ ((Cycle == Input) && (CurrentAirlock == AL2)) -> receiveDoorState(DI2,Closed)
.sendDoorRequest(DI2,Open).LampWaferHandler(CurrentAirlock = AL2)

+ ((Cycle == Input) && (CurrentAirlock == AL2)) -> receiveDoorState(DI2,Open)
.Move(R3,A2).PickupWafer(R3,A2).Move(R3,Lamp).PlaceWafer(R3,Lamp)
.LampWaferHandler(Cycle = Output)

+ ((Cycle == Output) && (CurrentAirlock == AL2)) -> CheckLamp-
State(Incomplete).LampWaferHandler(Cycle = Output)

+ ((Cycle == Output) && (CurrentAirlock == AL2)) -> CheckLamp-
State(Complete).Move(R3,Lamp).PickupWafer(R3,Lamp).Move(R3,A2).PlaceWafer(R3,A2)
.sendWaferStatus(AL2,Finished).LampWaferHandler(Cycle = Input, CurrentAir-
lock = None);

init

allow(
Move,
PickupWafer,
PlaceWafer,
CheckIPStackState,
CheckOPStackState,
CheckLampState,
OpenDoor,
CloseDoor,

commError,
commDoorState,
commDoorRequest,
commWaferStatus,
commWaferPresence,

comm(
receiveDoorState | sendDoorState -> commDoorState,
receiveDoorRequest | sendDoorRequest -> commDoorRequest,

receiveWaferStatus | sendWaferStatus -> commWaferStatus,
receiveWaferPresence | sendWaferPresence -> commWaferPresence
receiveError | sendError -> commError ,

IOHandler1(Get, Input) | | IOHandler2(Get, Input) | | AirlockChamber1Controller(NoWafer, Closed, Closed) | | AirlockChamber2Controller(NoWafer, Closed, Closed) | | LampWaferHandler(Input,None)
));

B Code: μ-Calculus

1. N.A.

2. mcrl2 MCF:
   (a) [true* . PickupWafer(R1, I1) . (!PlaceWafer(R1,O1))*.
   PickupWafer(R1,I1)] false
   (b) [true* . PickupWafer(R2, I2) . (!PlaceWafer(R2,O2))*.
   PickupWafer(R2,I2)] false

3. mcrl2 MCF:
   (a) [true* . PlaceWafer(R1, I1)] false
   (b) [true* . PlaceWafer(R2, I2)] false

4. mcrl2 MCF:
   (a) [true* . CheckIPStackState(IP1, Empty).
   (!CheckIPStackState(IP1, NonEmpty))*. PickupWafer(R1, I1)] false
   (b) [true* . CheckIPStackState(IP2, Empty).
   (!CheckIPStackState(IP2, NonEmpty))*. PickupWafer(R2, I2)] false

5. mcrl2 MCF:
   (a) [true*.PickupWafer(R1,I1)] <true*.PlaceWafer(R1,A1)> true
   (b) [true*.PickupWafer(R2,I2)] <true*.PlaceWafer(R2,A2)> true

6. mcrl2 MCF:
   (a) [(!commWaferPresence(AL1,Finished))*.PickupWafer(R1,A1)] false
   (b) [(!commWaferPresence(AL2,Finished))*.PickupWafer(R2,A2)] false

7. mcrl2 MCF:
   [true*] forall d : DoorID.[OpenDoor(d).(!CloseDoor(d))*.
   OpenDoor(CorrespondingDoor(d))] false

8. mcrl2 MCF:
   (a) [true*.PickupWafer(R3,A1)] <true*.PlaceWafer(R3,Lamp)> true
   (b) [true*.PickupWafer(R3,A2)] <true*.PlaceWafer(R3,Lamp)> true

9. mcrl2 MCF:
   [!CheckLampState(Complete)*.PickupWafer(R3,Lamp)] false

10. mcrl2 MCF:
    [true*.PickupWafer(R3,Lamp).!(PlaceWafer(R3,A1) || PlaceWafer(R3,A2))*.
    PlaceWafer(R3,Lamp)] false

11. mcrl2 MCF:
    (a) [true*.PickupWafer(R3,A1).!(PickupWafer(R3, Lamp))*.
    PickupWafer(R3, Lamp)] <true*.(PlaceWafer(R3, A1))> true
    (b) [true*.PickupWafer(R3,A2).!(PickupWafer(R3, Lamp))*.
    PickupWafer(R3, Lamp)] <true*.(PlaceWafer(R3, A2))> true

12. mcrl2 MCF:
    [true* . receiveWaferStatus(AL1, Finished). true*.
    PickupWafer(R3, A1)] false

13. mcrl2 MCF:
    (a) [true*.PickupWafer(R1, A1)] <true*.PlaceWafer(R1,O1)> true
    (b) [true*.PickupWafer(R2, A2)] <true*.PlaceWafer(R2,O2)> true

14. mcrl2 MCF:
    (a) [true* . PickupWafer(R1, O1)] false
    (b) [true* . PickupWafer(R2, O2)] false

15. mcrl2 MCF:
    (a) [true* .CheckOPStackState(OP1,Full).
    (!CheckOPStackState(OP1, NonFull))* . PlaceWafer(R1, O1)] false
    (b) [true* .CheckOPStackState(OP2,Full).
    (!CheckOPStackState(OP2, NonFull))* . PlaceWafer(R2, O2)] false

16. mcrl2 MCF:
    (a) [true* .PickupWafer(R1, I1) .(!PickupWafer(R1, A1))* .
    PlaceWafer(R1, O1)] false
    (b) [true* .PickupWafer(R2, I2) .(!PickupWafer(R2, A2))* .
    PlaceWafer(R2, O2)] false

17. mcrl2 MCF:
    [true*].<true>.true