System Validation Project Report

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 10, 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 formulae. 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 -calculus formulae. 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: A1, A2

- Outer Robots: R1, R2

- Inner Robot: R3

2.2 System Requirements

The behaviour of the system can be understood by describing the individual components requirements.

1. The Robots (R1 and R2) should not move to the Input Stacks if the Input Stacks are Empty.

2. The Robots (R1 and R2) should not move to the Output Stacks if the Output Stacks are Full.

3. The Robots (R1 and R2) should not place a new wafer on the Output Stacks (O1 and O2).

4. The Inner Door (DI1) must not be opened if the Outer Door(DO1) is open for Airlock (A1).

5. The Inner Door (DI2) must not be opened if the Outer Door(DO2) is open for Airlock (A2).

6. The Outer Door (DO1) must not be opened if the Inner Door(DI1) is open for Airlock (A1).

7. The Outer Door (DO2) must not be opened if the Inner Door(DI2) is open for Airlock (A2).

8. The Inner Doors (DI1 and DI2) must not be opened if a finished wafer is present in their corresponding Airlocks (A1 and A2).

9. The Outer Doors (DO1 and DO2) must not be opened if a new wafer is present in their corresponding Airlocks (A1 and A2).

10. The Robot (R3) will pickup the finished wafer from the Lamp (L) only when it is finished printing.

11. As long as the wafer can move through the production process, it will. In other words, the system is deadlock free.

3 Interactions

3.1 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)**

- MoveTo(x) [x: DestinationID] : Move to assigned destination.

- PickupWafer : Picks up the wafer.

- PlaceWafer : Places the wafer.

- OpenDoor(x) [x: DI1, DI2, DO1, DO2] : Opens the corresponding door.

- CloseDoor(x) [x: DI1, DI2, DO1, DO2] : Closes the corresponding door.

The commands 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] :

3.2 Communications

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

Command(ComponentID, State):

- receiveDoorState(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- sendDoorState(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- commDoorState(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- receiveDoorRequest(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- sendDoorRequest(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- commDoorRequest(x,s) [x: DI1, DI2, DO1, DO2 ; s: Open, Closed]

- receiveWaferStatus(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

- sendWaferStatus(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

- commWaferStatus(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

- receiveWaferPresence(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

- sendWaferPresence(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

- commWaferPresence(x,s) [x: A1, A2 ; s: New, Finished , NoWafer]

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.
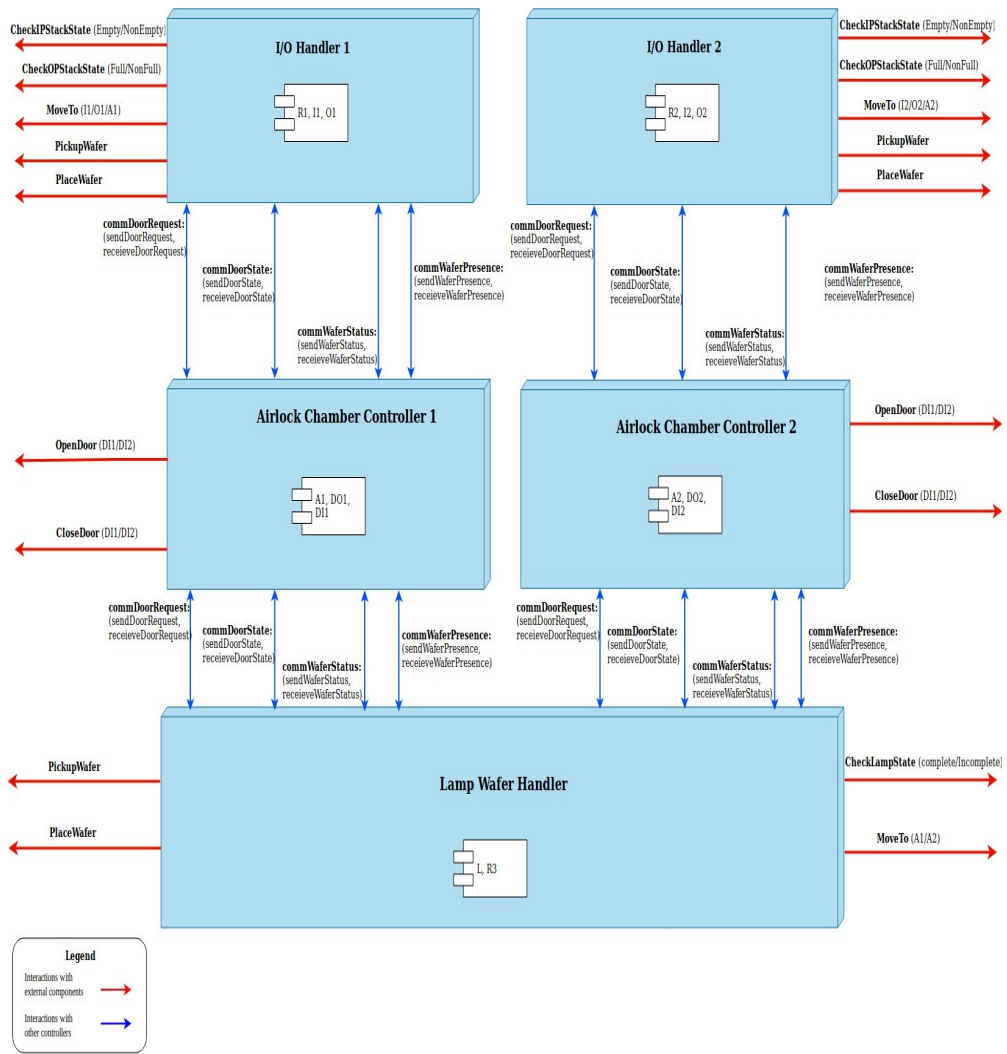
Figure 1: Architecture Diagram of System

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 -formulae

Here the corresponding Modal μ -formulae are described:

1. The Robots (R1 and R2) should not move to the Input Stacks if the Input Stacks are Empty:

   Formula: $[true*]\forall IP : IPStackID.[CheckIPStackState(IP, Empty).$
   $(\overline{MoveTo(MapIPDestination(IP)}*].true$
   Status: Verified true.

2. The Robots (R1 and R2) should not move ohe Output Stacks if the Output Stacks are Full:

   Formula: $[true*]\forall OP : OPStackID.[CheckOPStackState(OP, Full).$
   $(\overline{MoveTo(MapOPDestination(OP)}*].true$
   Status: Verified true.

3. The Robots (R1 and R2) should not place a new wafer on the Output Stacks (O1 and O2):

   - **Formula:** $[true*].[MoveTo(I1).(\overline{MoveTo(AL1}*.MoveTo(O1)].false$
     Status: Verified true.
   - **Formula:** $[true*].[MoveTo(I2).(\overline{MoveTo(AL2}*.MoveTo(O2)].false$
     Status: Verified true.

4. The Inner Door (DI1) must not be opened if the Outer Door(DO1) is open for Airlock (A1).

5. The Inner Door (DI2) must not be opened if the Outer Door(DO2) is open for Airlock (A2).

6. The Outer Door (DO1) must not be opened if the Inner Door(DI1) is open for Airlock (A1).

7. The Outer Door (DO2) must not be opened if the Inner Door(DI2) is open for Airlock (A2) : **(for 4-7)**

   Formula: $[true*]\forall d : DoorID.[OpenDoor(d).\overline{CloseDoor(d)}*OpenDoor(Corresponding(d)$

   Status: Verified true.

8. The Inner Doors (DI1 and DI2) must not be opened if a finished wafer is present in their corresponding Airlocks (A1 and A2):

   - **Formula:** $[true*.receiveWaferStatus(AL1, Finished).(\overline{OpenDoor(DI1)}]true$
     Status: Verified true.

   - **Formula:** $[true*.receiveWaferStatus(AL2, Finished).(\overline{OpenDoor(DI2)}]true$
     Status: Verified true.

9. The Outer Doors (DO1 and DO2) must not be opened if a new wafer is present in their corresponding Airlocks (A1 and A2):

   - **Formula:** $[true*.receiveWaferStatus(AL1, New).(\overline{OpenDoor(DO1)}]true$
     Status: Verified true.

   - **Formula:** $[true*.receiveWaferStatus(AL2, New).(\overline{OpenDoor(DO2)}]true$
     Status: Verified true.

10. The Robot (R3) will pickup the finished wafer from the Lamp (L) only when it is finished printing:

    Formula: $[true*.CheckLampState(Incomplete).(\overline{MoveTo(Lamp}*].true$
    Status: Verified true.

11. The system is deadlock free:

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

6 Modelling the System

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. The system currently has 87 levels, 1740 states and 3776 transitions.

We have however, restricted 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 (A1 or A2) 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 that 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 modelling a system which will have the flexibility to move the robots to different stacks and hence increase the throughput.
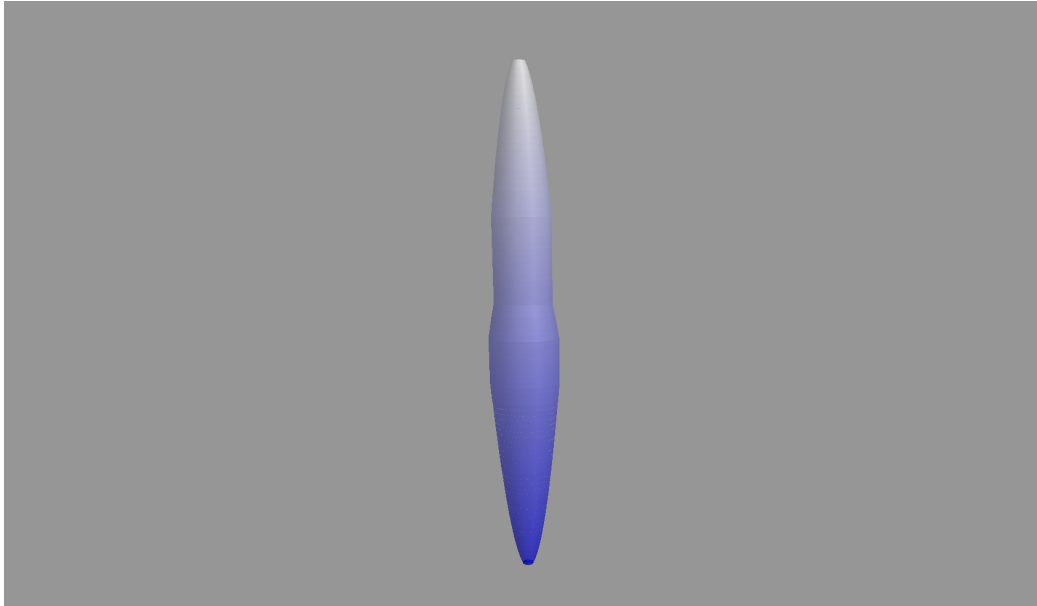

Figure 2: LTSView of the System

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 for the complete system we simply needed to extend from this simplification by doubling the number of components mentioned above. The Complete Model was visually inspected using LTSView to mark for deadlocks along with an LPSxSim simulation to logically verify the correct sequence of actions for the process. These are presented in the Visualization subsection. The μ formulae 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 : Editing

- mcrl22lps : Transformation

- lpsxsim : Simulation

- lps2lts : Transformation

- ltsgraph : Visualization

- ltsview : Visualization

- lts2pbes : Transformation

- pbessolve : Verification

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 MCF individually. The MCF presented in Section 5 are verified individually and each of the presented MCF 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.
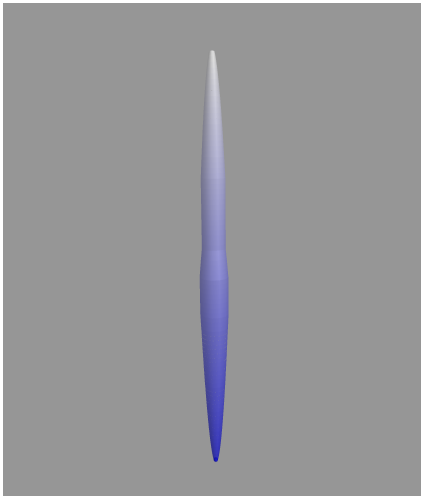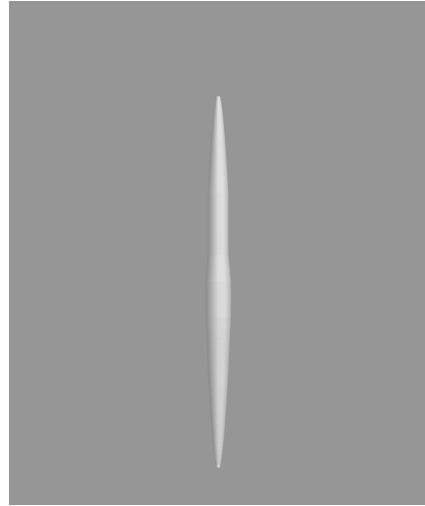

Figure 3: LTSView

Figure 4: Marking Deadlocks

8   Conclusion

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 formulae. In the fifth phase the model was verified by translating the μ-calculus formulae from phase four into mcf code. We were able to correctly specify the requirements in such a way that all translated requirements were met.

A Code: mcrl2

sort DestinationID = **struct** A1 | A2 | I1 | I2 | O1 | O2 | Lamp;
StackID = **struct** IP1 | IP2 | OP1 | OP2;
AirlockID = **struct** AL1 | AL2 | None;
IOHandlerID = **struct** IOH1 | IOH2;
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** New | 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 MoveTo: DestinationID;
PickupWafer;

PlaceWafer;

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(Ope
= Get)
+ ((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP1,NonEmpty).MoveTo(I
= Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO1,Closed).sendDoorReque
= Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO1,Open).MoveTo(A1).Pla
= Output, Operation = Get)

+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPres-
ence(AL1,NoWafer).IOHandler1(Operation = Get)
+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPres-
ence(AL1,Finished).receiveDoorState(DO1,Closed).sendDoorRequest(DO1,Open)
.receiveDoorState(DO1,Open).MoveTo(A1).PickupWafer.IOHandler1(Operation

15

= Put)

+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,Full).IOHandler1(O
= Put)
+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,NonFull).MoveTo(O
= Input, Operation = Get);

IOHandler2(Operation : OperationType, Cycle : CycleType) =
((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP2,Empty).IOHandler2(Ope
= Get)
+ ((Cycle == Input) && (Operation == Get)) -> CheckIPStackState(IP2,NonEmpty).MoveTo(I2
= Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO2,Closed).sendDoorReque
= Put)
+ ((Cycle == Input) && (Operation == Put)) -> receiveDoorState(DO2,Open).MoveTo(A2).Pla
= Output, 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)
.receiveDoorState(DO2,Open).MoveTo(A2).PickupWafer.IOHandler2(Operation
= Put)

+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP2,Full).IOHandler2(O
= Put)
+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP2,NonFull).MoveTo(O
= Input, Operation = Get);

AirlockChamber1Controller(WaferPresence : WaferType, OuterDoorState :
DoorState, InnerDoorState : DoorState) = ((OuterDoorState == Closed)
&& (InnerDoorState == Open)) -> receiveDoorRequest(DO1,Open).AirlockChamber1Controller(
= 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-

16

ceiveDoorRequest(DI1,Open).OpenDoor(DI1).AirlockChamber1Controller(InnerDoorState = Open)

+ ((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 = 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 == 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)) -> receiveWaferPres-
ence(AL1,New).LampWaferHandler(CurrentAirlock = AL1)
+ ((Cycle == Input) && (CurrentAirlock == AL1)) -> receiveDoorState(DI1,Closed).sendDoorR
= AL1)
+ ((Cycle == Input) && (CurrentAirlock == AL1)) -> receiveDoorState(DI1,Open).MoveTo(A1
= Output)

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

+ ((Cycle == Output) && (CurrentAirlock == AL1)) -> CheckLamp-
State(Complete).MoveTo(Lamp).PickupWafer.MoveTo(A1).PlaceWafer.sendWaferStatus(AL1,Fi
= Input, CurrentAirlock = None)

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

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

+ ((Cycle == Input) && (CurrentAirlock == AL2)) -> receiveDoorState(DI2,Open).MoveTo(A2
= Output)

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

+ ((Cycle == Output) && (CurrentAirlock == AL2)) -> CheckLamp-
State(Complete).MoveTo(Lamp).PickupWafer.MoveTo(A2).PlaceWafer.sendWaferStatus(AL2,Fi
= Input, CurrentAirlock = None);

init

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

commDoorState,
commDoorRequest,
commWaferStatus,
commWaferPresence,

comm(
receiveDoorState | sendDoorState -> commDoorState,
receiveDoorRequest | sendDoorRequest -> commDoorRequest,
receiveWaferStatus | sendWaferStatus -> commWaferStatus,
receiveWaferPresence | sendWaferPresence -> commWaferPresence,

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

B Code: μ-formulae

1. mcrl2 MCF: [true*] forall IP : IPStackID . [CheckIPStackState(IP,Empty). (!MoveTo(MapIPDestination(IP)))*] true

2. mcrl2 MCF: [true*] forall OP : OPStackID . [CheckOPStackState(OP,Full). (!MoveTo(MapOPDestination(OP)))*] true

3. mcrl2 MCF: (a) [true*] [MoveTo(MapIPDestination(IP1)). (!MoveTo(MapAirlock(AL1)))* .MoveTo(MapOPDestination(OP1))] false

   (b) [true*] [MoveTo(MapIPDestination(IP2)). (!MoveTo(MapAirlock(AL2)))*.MoveTo(Map false

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

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

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

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

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

9. mcrl2 MCF: [true*.receiveWaferStatus(AL1, Finished). !OpenDoor(DI1)*] true

10. mcrl2 MCF: [true*.receiveWaferStatus(AL1, New). !OpenDoor(DO1)*] true

11. mcrl2 MCF: [true*].¡true¿.true