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 Formulas. 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 move to the Output Stacks without a finished wafer.
4. The Robots (R1 and R2) should not place a new wafer on the Output Stacks (O1 and O2).
5. The Robots (R1 and R2) should not place a finished wafer on the Input Stacks (I1 and I2).
6. The Robots (R1 and R2) should not move to the Airlocks (A1 and A2) if the corresponding Outer Doors are closed (DO1 and DO2).
7. The Robot (R3) should not move to the Airlocks (A1 or A2) if the corresponding Inner Doors are closed.
8. The Inner Door (DI1) must not be opened if the Outer Door(DO1) is open for Airlock (A1).
9. The Inner Door (DI2) must not be opened if the Outer Door(DO2) is open for Airlock (A2).
10. The Outer Door (DO1) must not be opened if the Inner Door(DI1) is open for Airlock (A1).
11. The Outer Door (DO2) must not be opened if the Inner Door(DI2) is open for Airlock (A2).
12. The Inner Doors (DI1 and DI2) must not be opened if a finished wafer is present in their corresponding Airlocks (A1 and A2).
13. The Outer Doors (DO1 and DO2) must not be opened if a new wafer is present in their corresponding Airlocks (A1 and A2).
14. The Robot (R3) will place the new wafer on the Lamp only when it is empty.

15. The Robot (R3) will pickup the finished wafer from the Lamp (L) only when it is finished printing.
16. The Robot (R3) will not place the finished wafer again on the Lamp (L).
17. The Robot (R3) will not pickup a wafer immediately after it has placed a wafer.
18. The Robot (R3) will place a wafer it picked from an Airlock (A1 or A2) only to the same Airlock (after processing).

2.3 Functional Requirements

2.3.1 Overall requirements of the System:

- As long as the Output Stacks are not full, wafers keep moving along the production line. (System is deadlock free) **(Safety)**
- Both the Inner and Outer Doors of an Airlock must NOT be open at the same time. **(Safety)**
- Doors must not be closed while the robot is picking up/placing a wafer in the Airlock **(Safety)**

2.3.2 System Component Requirements:

These are the individual component requirements:

1. Doors: DI1, DI2, DO1, DO2
   - Inner and Outer Doors (DIx, DOx) of an Airclock (Ax) should never be open at the same time. (Vaccuum must be preserved) **(Safety)**
   - Inner Door (DIx) should be opened ONLY if it's corresponding Outer Door (DOx) is closed and vice versa. **(Safety)**
2. Output Stacks: O1, O2
   - Wafer should not be put on a full Output Stack. **(Safety)**
3. Outer Robots: R1, R2
   - The Outer Robot (Rx) should pick the next Input wafer ONLY after delivering a finished wafer to it's corresponding Output Stack (Ox). **(Safety)**

4. Inner Robot: R3

    - The Inner Robot should place the Input wafer on the Lamp ONLY when the Lamp's stack is empty. **(Safety)**
    - The Inner Robot should pick the next Input wafer ONLY after delivering a finished wafer to it's corresponding Airlock (Ax). **(Safety)**

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 actuator, optional parameters)

- MoveTo(x, D) [x: R1, R2, R3 ; D: Lamp, Airlocks, Input/Output Stacks] Move to assigned destination.
- PickupWafer(x, D) [x: R1, R2, R3 ; D: Lamp, Airlocks, Input Stacks] Picks up the wafer.
- PlaceWafer(x, D) [x: R1, R2, R3 ; D: Lamp, Airlocks, Output Stacks] Places the wafer.
- OpenDoor(x,s) [x: DI1, DI2, DO1, DO2; s:Open, Close] 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	✓	✓	✓				

3.2 Communications

The following are the commands used by the controller to read the data provided by the sensors of the system. The meaning can be interpreted as:

Command(Target sensor, Return Value by sensor)

- CheckInputStack(x, s) [x: I1,I2 ; s: Present, Absent]
- CheckOutputStack(x, s) [x: O1,O2 ; s: Full, Empty]
- CheckLamp(L, s) [L: Lamp ; s: Busy, CompletedPrinting]
- CheckAirlock(x, s) [x: A1,A2 ; s : WaferPresent, WaferAbsent]

4 Architecture

4.1 Global System Architecture

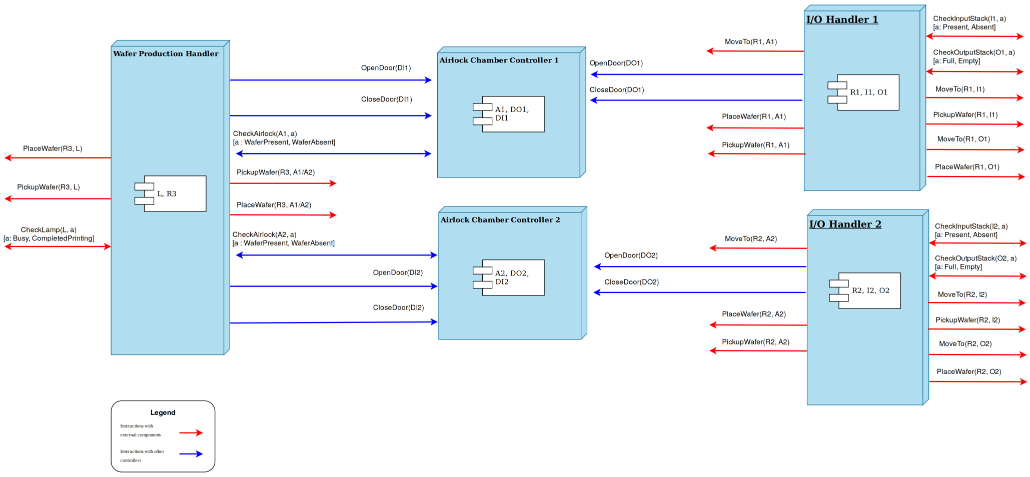

Figure 1: Architecture Diagram of System

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

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.

In order to translate the requirements into their corresponding MCF representations, the requirements are rephrased as follows:

1. Requirement 1
    - This is the rephrased version
    - May need to add more than one
2. Requirement 2
    - This is the rephrased version
    - May need to add more than one
3. Requirement 3
    - This is the rephrased version
    - May need to add more than one
4. Requirement 4
    - This is the rephrased version
    - May need to add more than one
5. Requirement 5
    - This is the rephrased version
    - May need to add more than one
6. Requirement 6
    - This is the rephrased version
    - May need to add more than one
7. Requirement 7

- This is the rephrased version
- May need to add more than one

8. Requirement 8
   - This is the rephrased version
   - May need to add more than one
9. Requirement 9
   - This is the rephrased version
   - May need to add more than one
10. Requirement 10
    - This is the rephrased version
    - May need to add more than one

5.2 Modal μ -Formulas

Here the corresponding Modal μ -Formulas are described:

1. Translated Requirement 1:
   - MCF
2. Translated Requirement 2:
   - MCF
3. Translated Requirement 3:
   - MCF
4. Translated Requirement 4:
   - MCF
5. Translated Requirement 5:
   - MCF
6. Translated Requirement 6:
   - MCF

7. Translated Requirement 7:
    - MCF
8. Translated Requirement 8:
    - MCF
9. Translated Requirement 9:
    - MCF
10. Translated Requirement 10:
    - MCF

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 ——— states and ——— 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.

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 logeically verify the correct sequence of actions for the process. These are presented in the Visualization subsection. The μ Formulas 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

The following tools were used for modelling and verification:

- mcrl2xi : Editing
- mcrl22lps : Transformation
- lpsxsim : Simulation
- lps2lts : Transformation
- ltsgraph : Visualization
- ltsview: Visualization

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

7.3 Visualizations

8 Conclusion

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;

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

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
= Put)

+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,Full).IOHandler1(C
= Put)
+ ((Cycle == Output) && (Operation == Put)) -> CheckOPStackState(OP1,NonFull).MoveTo(
= 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(I
= 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)) -> receiveWaferPresence(AL2,NoWafer).IOHandler2(Operation = Get)
+ ((Cycle == Output) && (Operation == Get)) -> receiveWaferPresence(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(
= Input, Operation = Get);

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

+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> receiveDoorRequest(DO1,Open).OpenDoor(DO1).AirlockChamber1Controller(OuterDoorState
= Open)
+ ((OuterDoorState == Open) && (InnerDoorState == Closed)) -> receiveDoorRequest(DI1,Open).AirlockChamber1Controller(OuterDoorState =
Open)
+ ((OuterDoorState == Closed) && (InnerDoorState == Closed)) -> receiveDoorRequest(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)) -> receiveWaferPresence(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)) -> CheckLampState(Incomplete).LampWaferHandler(Cycle = Output)

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

+ ((Cycle == Input) && (CurrentAirlock == None)) -> receiveWaferPresence(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-Formulas