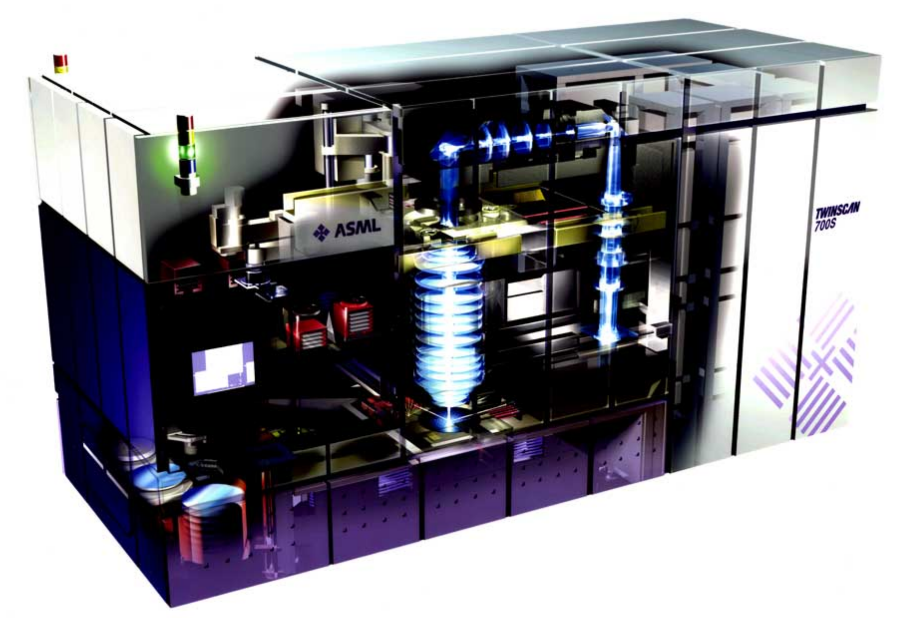
# System Validation of ASML wafer stepper

System Validation IN4387 Group 2

A. Delawari (anton.delawari@gmail.com)
C. Hartman (c.hartman@student.tudelft.nl)
Z. Kocsi-Horvath (z.kocsihorvath@student.tudelft.nl)
M. Slotema (m.slotema@student.tudelft.nl)

21st January 2012



## Contents

## 1 Introduction

For the next generation of the wafer stepper, ultraviolet light is used to project the desired layout onto the wafer surface. Due to the absorption of ultraviolet light by the atmosphere, the wafer stepper needs to operate in high vacuum. Furthermore, the lenses in the system are highly sensitive to atmospheric conditions, thus the time the lenses are exposed to normal air needs to be minimized. For economic reasons the speed at which the wafer stepper is operated should be maximised. These requirements dictate the use of a multi-chamber installation in order to maintain a constant high vacuum in the machine and to achieve a high throughput. The prototype of the multi-chamber system is depicted in Figure 1. It consists of two sluices, one low vacuum chamber and one high vacuum chamber. Each of the two sluices has two doors, an outer door and an inner door. Wafers move from normal atmospheric pressure into the sluice through the outer door, where the air pressure is lowered to match the level of the low vacuum chamber. Once the pressure of the sluice equals the low vacuum chamber, the inner door is opened and the wafer is transported into the low vacuum chamber. In the low vacuum chamber the pressure is further lowered to match the level of the high vacuum chamber. Afterwards, the wafers in the high vacuum chamber and low vacuum chamber can be exchanged. Wafers leaving the system will be transported in the reverse direction. Later versions might be extended with more sluices to increase the performance of the system.

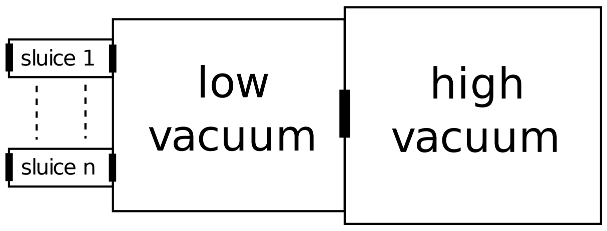


Figure 1: A brief overview of the wafer stepper.

In this document, the controller designed for such a wafer stepper is described. In the section 2 the global requirements are briefly discussed. Section 3 is dedicated to interactions which are translated from the requirements. The requirements from section 2 are translated into modal $\mu$-calculus in section 4. The global architecture is presented in section 5, afterwards each processor is discussed separately. The mCRL2 model of each controller is described in section 6. In section 7, the designed controllers are verified by showing that they match the given requirements. Furthermore, the changes made in the model or requirements in order to make them pass verification are explained. Finally, the conclusions will be presented in section 8.

## 2 Global requirements

The following global requirements were formulated:

1. A door may only be opened if the pressure on both sides of the door is equal.
2. A sluice may only have one door open.
3. When the door between the low vacuum chamber and the high vacuum chamber is open, all the doors between the low vacuum chamber and the sluices should be closed.
4. The high vacuum chamber may hold at most one wafer.
5. The low vacuum chamber may hold at most two wafers.
6. A sluice may hold at most one wafer.
7. It should always be possible to know the number of wafers currently in the system.
8. If the door between the low vacuum chamber and the high vacuum chamber malfunctions, the machine is broken. This should be indicated.
9. If a sluice door malfunctions, the other sluice(s) should be used instead.
10. If there is a wafer ready to enter (resp. leave) the system, it should eventually do so, unless the system shuts down.

## 3 External interactions

In this section, the external interactions are defined. External interactions are interactions that take place between the controller and the rest of the system.

Table 1 provides a list of all the external interactions, together with a short description and a list of the parameter types. The possible values a parameter type can take is supplied in table 2.

Table 1: List of external interactions

| Action | Description | Parameter |
|---|---|---|
| Equalise | Changes the air pressure to the indicated pressure level | Room, Pressure |
| ExitWafer | Removes one waver from the system | SluiceNR, Wafer |
| HighVac_From | Transfers a wafer from the High Vacuum Chamber to the Low Vacuum Chamber | Wafer |
| HighVac_To | Transfers a wafer from the Low Vacuum Chamber to the High Vacuum Chamber | Wafer |
| HighVacDoor | Opens or closes the High Vacuum door | DoorState |
| HighVacDoorState | Indicates if the door to the High Vacuum chamber is completely open | DoorState |
| InnerDoor_Sluice | Opens or closes the inner door of a Sluice | SluiceNR, DoorState |
| InnerDoorState_Sluice | Indicates the state of the inner door of a Sluice | SluiceNR, DoorState |
| InputWafer | Puts a wafer into the system | SluiceNR, Wafer |
| ManOpen_Sluice | Manually opens the outer door of a Sluice | SluiceNR |
| OuterDoor_Sluice | Opens or closes the outer door of a Sluice | SluiceNR, DoorState |
| OuterDoorState_Sluice | Indicates the state of the outer door of a Sluice | SluiceNR, DoorState |
| Pressure_HighVac | Pressure in the High Vacuum chamber | Pressure |
| Pressure_LowVac | Pressure in the Low Vacuum chamber | Pressure |
| Pressure_Sluice | Pressure in Sluice N | SluiceNR, Pressure |
| ReportStatus | Reports the number of wafers in the system to the 'user' | Nat |
| RoomFailure | Indicates if a room is broken | Room |
| RoomStatus | Reports the number of wafers in a room | Room, Nat |
| SluiceSync_comm | Indicates that a sluice has just closed its InnerDoor | SluiceNR |
| Status | Queries the number of wafers in the system from the controller | |
| ToLowVac_Comm | Transfers a wafer from a sluice to the Low Vacuum Chamber | SluiceNR, Wafer |
| ToSluice_Comm | Transfers a wafer from the Low Vacuum Chamber to a sluice | SluiceNR, Wafer |
| WantIn | The World want to insert a wafer into the sluice | SluiceNR |
| WantOut | Want to output wafer from sluice into the World | SluiceNR |

Table 2: List of external datatypes

| Data type | Values |
|---|---|
| SluiceNR | One, Two |
| Pressure | Atm, LowVacuum or HighVacuum |
| DoorState | Open or Close |
| Wafer | |
| Nat | Natural number |
| Room | LowVac, HighVac, SluiceOne, SluiceTwo |

## 4 Translated requirements

This section lists the requirements from section 2 in terms of the interactions from section 3.

1.
    - Before the outer doors can be opened the pressure in the sluice should be equalized to atmospheric pressure:
      $\forall$ Sl $\in$ Sluices : [ ( ( !Equalise(Sl, Atm) )* . Equalise(Sl, Atm) . ( !OuterDoor_Sluice(One, Open) && !Equalise(Sl, LowVacuum) && !Equalise(Sl, HighVacuum) )* . ( Equalise(Sl, LowVacuum) + Equalise(Sl, HighVacuum) ) . OuterDoor_Sluice(Sl, Open) ) + ( (!Equalise(Sl, Atm) )*. OuterDoor_Sluice(Sl, Open) ) ]false
    - An inner door can only be opened when the pressure in the sluice and in the Low Vacuum chamber is at Low Vacuum:
      $\forall$ Sl $\in$ Sluices : [ true* . ( ( Equalise(Sl, Atm) . !Equalise(Sl, LowVacuum)* ) + ( Equalise(LowVac, HighVacuum) . !Equalise(LowVac, LowVacuum)* ) ) . InnerDoor_Sluice(Sl, Open) ] false
    - Before the highvacuum door can be opened the pressure in the LowVacuum room should be equalized to highvacuum pressure:
      [ ( ( !Equalise(LowVac, HighVacuum) )* . Equalise(LowVac, HighVacuum) . ( !HighVacDoor(Open) && !Equalise(LowVac, LowVacuum) && !Equalise(LowVac, Atm) )* . ( Equalise(LowVac, LowVacuum) + Equalise(LowVac, Atm) ) . HighVacDoor(Open) ) + ( ( !Equalise(LowVac, HighVacuum) )* . HighVacDoor(Open) ) ]false
2.
    - If the outer door is open, the inner door cannot be opened:
      $\forall$ Sl $\in$ Sluices : [ true* . OuterDoor_Sluice(Sl, Open) . OuterDoorState_Sluice(Sl, Open) . !OuterDoor_Sluice(Sl, Close )* . InnerDoor_Sluice(Sl, Open) ] false
    - If the inner door is open, the outer door cannot be opened: $\forall$ Sl $\in$ Sluices : [ true* . InnerDoor_Sluice(Sl, Open) . InnerDoorState_Sluice(Sl, Open) . !InnerDoor_Sluice(Sl, Close )* . OuterDoor_Sluice(Sl, Open) ] false
3.
    - If the HighVacuum door is open, no sluice doors can be opened:
      $\forall$ Sl $\in$ Sluices : [ !HighVacDoor(Open)* . HighVacDoor(Open) . ( ( !HighVacDoor(Close) && !InnerDoor_Sluice(One, Open) )* + HighVacDoor(Close) . !HighVacDoor(Open)* . HighVacDoor(Open) )* . InnerDoor_Sluice(One, Open) ] false
    - The HighVacuum door can only be opened if all sluice doors are closed:

∀ Sl ∈ Sluices : [ true* . InnerDoor_Sluice(Sl, Open) . !Inner-Door_Sluice(Sl, Close)* . HighVacDoor(Open) ] false

4. 
   - There is a wafer in the High Vacuum chamber, so no wafer is allowed to enter:
     [ true* . RoomStatus(HighVac, 1) ] $\mu X$ . ( [ !( HighVac_From(Wf) && HighVac_To(Wf) ) ]$X$ || < HighVac_From(Wf) > true || [ HighVac_To(Wf) ] false )
   - There are no wafers in the High Vacuum chamber, so no wafer is allowed to leave:
     [ true* . RoomStatus(HighVac, 0) ] $\mu X$ . ( [ !( HighVac_From(Wf) && HighVac_To(Wf) ) ]$X$ || < HighVac_To(Wf) > true || [ High-Vac_From(Wf) ] false )

5. 
   - The Low Vacuum chamber holds two wafers, so no wafer is allowed to enter:
     ∀ Sl ∈ Sluices : [ true* . RoomStatus(LowVac, 2) ] $\mu X$ . ( [ !( ToSluice_Comm(One, Wf) && ToLowVac_Comm(One, Wf) && HighVac_To(Wf) && HighVac_From(Wf) ) ] $X$ || < ToSluice_Comm(Sl, Wf) + HighVac_To(Wf) > true || [ ToLowVac_Comm(Sl, Wf) + High-Vac_From(w) ] false )
   - The Low Vacuum chamber holds no wafers, so no wafer is allowed to leave:
     ∀ Sl ∈ Sluices : [ true* . RoomStatus(LowVac, 2) ] $\mu X$ . ( [ !( ToSluice_Comm(One, Wf) && ToLowVac_Comm(One, Wf) && HighVac_To(Wf) && HighVac_From(Wf) ) ] $X$ || < ToLow-Vac_Comm(Sl, Wf) + HighVac_From(Wf) > true || [ ToSluice_Comm(Sl, Wf) + HighVac_To(w) ] false

6. 
   - There is a wafer in the sluice, so no wafer is allowed to enter:
     ∀ Sl ∈ Sluices : [ true* . ( InputWafer(Sl, Wf) + ToSluice_Comm(Sl, Wf) ) ] $\mu X$ . ( [ !( ExitWafer(Sl, Wf) && ToLowVac_Comm(Sl, Wf) && InputWafer(Sl, Wf) && ToSluice_Comm(Sl, Wf) ) ] $X$ || < ExitWafer(Sl, Wf) && ToLowVac_Comm(Sl, Wf) > true || [ In-putWafer(Sl, Wf) && ToSluice_Comm(Sl, Wf) ] false )
   - There are no wafers in the sluice, so no wafer is allowed to leave:
     ∀ Sl ∈ Sluices : [ true* . ( ExitWafer(Sl, Wf) + ToLowVac_Comm(Sl, Wf) ) ] $\mu X$ . ( [ !( InputWafer(Sl, Wf) && ToSluice_Comm(Sl, Wf) && ExitWafer(Sl, Wf) && ToLowVac_Comm(Sl, Wf) ) ] $X$ || < InputWafer(Sl, Wf) && ToSluice_Comm(Sl, Wf) > true || [ Ex-itWafer(Sl, Wf) && ToLowVac_Comm(Sl, Wf) ] false )

7. Status should eventually be possible and respond with a ReportStatus() action.

[ ( !Status )* . Status . ($\forall x \in Nat$ ( !ReportStatus(x) ) )* ] false

8. - The door should be open, but it is not:
     [ true* . HighVacDoor(Open) . !HighVacDoor(Close)* . HighVacDoorState(Close) . !RoomFailure(HighVac) ] false
   - The door should be closed, but it is not:
     [ true* . HighVacDoor(Close) . !HighVacDoor(Open)* . HighVacDoorStatus(Open) . !RoomFailure(HighVac) ] false

9. For every sluice:

   - The inner door should be open, but it is not:
     $\forall$ Sl $\in$ Sluices : [ true* . InnerDoor_Sluice(Sl, Open) . !InnerDoor_Sluice(Sl, Close)* . InnerDoorState_Sluice(Sl, Close) ] $\mu$ X . ( [ !RoomFailure(Sl) ] X && < RoomFailure(Sl) > true
   - The inner door should be closed, but it is not:
     $\forall$ Sl $\in$ Sluices : [ true* . InnerDoor_Sluice(Sl, Close) . !InnerDoor_Sluice(Sl, Open)* . InnerDoorState_Sluice(Sl, Open) . !RoomFailure(Sl) ] false
   - The outer door should be open, but it is not:
     $\forall$ Sl $\in$ Sluices : [ true* . OuterDoor_Sluice(Sl, Open) . !OuterDoor_Sluice(Sl, Close)* . OuterDoorStatus_Sluice(Sl, Close) . !RoomFailure(Sl) ] false
   - The outer door should be closed, but it is not:
     $\forall$ Sl $\in$ Sluices : [true* . OuterDoor_Sluice(Sl, Close) . !OuterDoor_Sluice(Sl, Open)* . OuterDoorState_Sluice(Sl, Open) . !RoomFailure(Sl) ] false

10. - If InputWafer( Wf ) and not RoomFailure( Rm ) $\forall$ Rm $\in$ Rooms then InputWaver( Wf ) should eventually be executed.
      $\forall$ Sl $\in$ Sluices : [ true* . WantIn(Sl) ] $\mu X$ . ( [ !( ( RoomFailure(SluiceOne) || RoomFailure(SluiceTwo) || RoomFailure(LowVac) || RoomFailure(HighVac) ) —— InputWafer(Sl, Wf) ) ] $X$ && < true > true )
    - After InputWafer( Wf ) and if not RoomFailure( Rm ) $\forall$ Rm $\in$ Rooms then ExitWafer( Wf ) shoud eventually be executed.
      $\forall$ Sl $\in$ Sluices : [ true* . InputWafer(Sl, Wf) ] $\mu X$ . ( [ !( ( RoomFailure(SluiceOne) || RoomFailure(SluiceTwo) || RoomFailure(LowVac) || RoomFailure(HighVac) ) || ExitWafer(Sl, Wf) ) ] $X$ && < true > true )

## 5 Architecture

The wafer stepper controller is decomposed into two parallel controllers, the sluice controller and the inner controller. The inner controller controls the low and high vacuum chambers and the sluice controller controls the sluice. As there is one sluice controller for every sluice, extensions of the system can easily be modelled by adding extra sluice controllers to the model. Figure 2 shows an overview of the decomposition into the two controllers, and table 3 lists the internal actions that are needed for communication between the two controllers.

Table 3: List of internal interactions

| Action | Description | Parameter |
|---|---|---|
| ToLowVac_Send | Sluice sends wafer to low vacuum chamber | Wafer |
| ToLowVac_Reciev | Low Vacuum chamber receives wafer from sluice | Wafer |
| ToLowVac_Comm | Communication between sluice and low vacuum chamber to send wafer to low vacuum chamber | Wafer |
| ToSluice_Send | Low Vacuum sends Wafer to Sluice | SluiceNR, Wafer |
| ToSluice_Recieve | Sluice receives wafer from the low vacuum chamber | SluiceNR, Wafer |
| ToSluice_Comm | Communication between sluice and low vacuum chamber to send wafer to low vacuum chamber | SluiceNR, Wafer |

## 6 Modelling the controller

To make development of the model easier and to allow multiple people working on the model at the same time, the model of the wafer stepper controller has been divided in to four files.

- globals.mcrl2: contains all the global actions and data types.
- init.mcrl2: initializes the different controllers and renames the communication actions.
- lowVacuum.mcrl2: model of the controller of the low and high vacuum chamber.
- sluice.mcrl2: model of the sluice controller.

Unfortunately MCRL2 does not support any way to include or combine files. To overcome this issue a simple script file that merges the different files into a single file was created. The resulting file is used for analysis of the complete model.

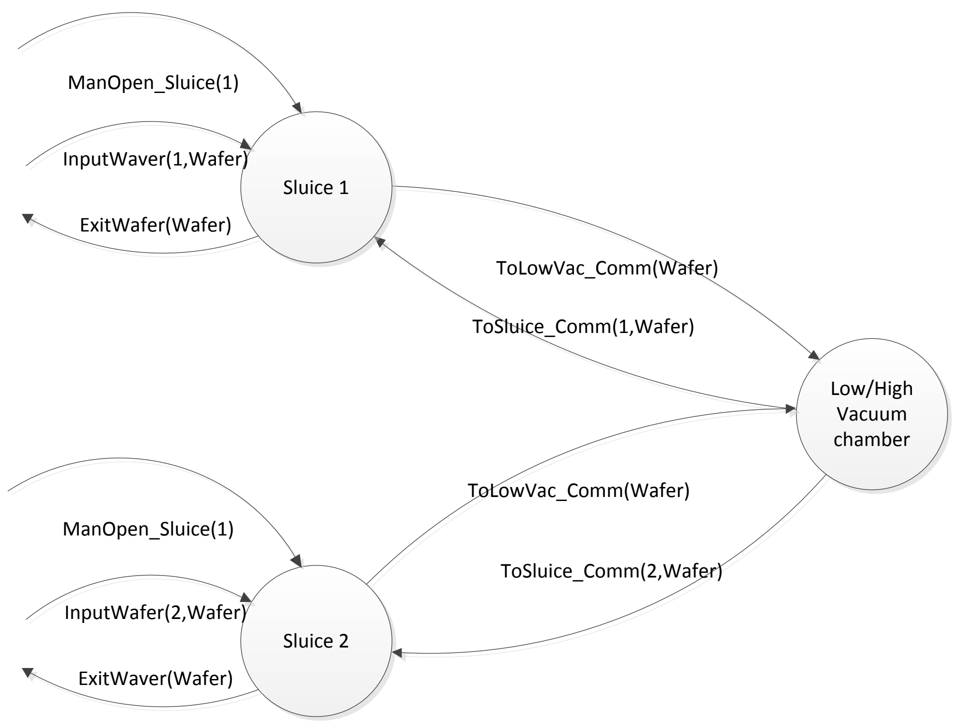


Figure 2: Overview of the decomposition to sluice and inner controller.

## 6.1 The sluice model

In this section the model of the sluice controller is described. It has three state variables:

- NR: the identification number of the sluice
- In: indicates whether or not a wafer from outside is ready to be inserted into the LowVacuum chamber through the sluice.
- Out: indicates that a processed wafer should be removed from the LowVacuum chamber through the sluice and be removed from the wafer stepper.

The initial state of these variables is [ID of sluice, false, false]. The first variable has been already set because the ID of the sluice should be given before the system is started. False of In and Out indicates that no wafers are waiting for sluice actions after initialization.

If there is a wafer waiting outside to be processed (`WantIn(NR)`), the sluice state variable `In` changes to true (`Sluice(NR,true,In)`). The same holds for the case when a wafer should be removed from the Low-Vacuum chamber and be brought to the outside world (`WantOut(NR)`), but then for the variable `Out` (`Sluice(NR,In,true)`). If the variable `In` is `true` and the LowVac controller is ready to accept a wafer, the outer door of the sluice opens (`OuterDoor_Sluice(NR,Open)`), the wafer enters the sluice (`InputWafer(NR,W)`), the outer door closes (`OuterDoor_Sluice(NR,Close)`), the pressure is set to the pressure of LowVacuum chamber (`Equalise(sluice2room(NR),LowVacuum)`), the inner door opens (`InnerDoor_Sluice(NR,Open)`), the wafer is moved to the `LowVacuum` chamber (`ToLowVac_Send( NR, W )`), the inner door closes (`InnerDoor_Sluice(NR,Close)`) and the last action for this sequence is equalizing the pressure in the sluice to atmospheric pressure (`Equalise(sluice2room(NR),Atm)`). When this is finished, the state variable `In` is set back to false `(Sluice(NR,false,Out)`).

The same holds for getting wafer from the LowVacuum chamber to outside but in reversed sequence.

## 6.2 The low and high vacuum chamber model

The model described in the lowVacuum.mcrl2 file is the model of the controller responsible for accepting wafers from the sluice, moving them to the high vacuum chamber, processing them and sending the processed wafer back through one of the sluices out of the machine. This model uses five state variables:

- LowVacStatus: indicates the number of wafers in the low vacuum chamber (zero, one or two).
- HighVacuumStatus: true if there is a wafer in the high vacuum chamber, false otherwise.
- Unprocessed: a list of unprocessed wafers in the system.
- Processed: a list of processed wafers in the system.
- HighVacuumContent: a list of wafers in the high vacuum chamber.

If there is an unprocessed wafer in the low vacuum chamber and the high vacuum chamber is empty, a wafer can be moved from the low to the high vacuum chamber by first equalizing the pressure in the low vacuum chamber to high vacuum, then opening the door to the high vacuum chamber and moving the wafer to the high vacuum chamber. Finally, the controller is restarted with the updated state of the system: one less wafer in the low vacuum chamber and a wafer in the high vacuum chamber. When there is a processed wafer in the high vacuum chamber and there is room in the low vacuum chamber, the wafer can be moved from the high vacuum chamber to the low vacuum chamber by first equalizing the pressure in the low vacuum chamber to high vacuum, then opening the high vacuum door and removing the wafer. Finally the controller is restarted with the updated states: no wafer in the high vacuum chamber and an extra wafer in the low vacuum chamber.

Wafers in the low vacuum room are either processed or unprocessed wafers. Processed wafers are moved out of the system by signalling the sluice controller that a wafer needs to be moved out of the system. The sluice controller will then open the door to the sluice and the LowVac controller will move the wafer into the sluice. The same process is being initiated when there is room in the low vacuum chamber: the low vacuum controller will signal a sluice that it is ready to accept a wafer into the system. Once the wafer is moved in or out of the system, the controller will restart and update the state variables.

### 6.3 Major changes to the model

In order to satisfy requirements eight and nine, some major changes were made to the model. A state variable RoomFailure was added to both controllers. After each action that manipulates a door, the sensor belonging to the door is checked. For example after InnerDoor(Close) in a sluice, there is a choice between InnerDoorState(Close) or InnerDoorState(Open). If it is open, the state variable is set to true and the controller jumps to the begin. This variable is checked at the beginning of each controller. If the variable is true, the action RoomFailure is executed and the controller jumps again to the beginning.

## 7 Verification

For model verification all the modal $\mu$-calculus formulae, as described in Section 4, are converted to separate $\mu$-calculus formulae (`.mcf`) files. With these requirements translated, it is possible to use the lps2pbes and pbes2bool tools to check whether or not the wafer stepper model satisfies all the specified requirements. To simplify checking these requirements, two batch-files were made to check all the mcf files automatically and print the results.

Due to the national evolution process of this project, the system model and the requirements that were formulated in the early phase of the project are quite different from the final ones. The mcf files used for verification had to be changed in sync with the simultaneous changing of the system model and requirements.

The problems encountered during the verification process can roughly be sorted in three different categories:

1. Errors due to incorrect modal $\mu$-formulae.
2. Errors due to unimplemented actions.
3. Errors due to a wrong model.

These three different categories will now be discussed separately.

## 7.1 Errors due to incorrect modal $\mu$-formulae

During testing, some of the requirements were met by the model, only not recognized as such by the pbes2bool program. For example the first part of requirement four. Based on the toolchain, the requirement was not met. Figure 3 shows the reduced state space for this requirement. Green states indicate that there is a wafer in the High Vacuum chamber, light blue states indicate that there is no wafer in the High Vacuum chamber, red actions insert a wafer into the High Vacuum chamber and dark blue actions remove a wafer from the High Vacuum chamber. Manual inspection of this model indicated that the requirement (when there is a wafer in the High Vacuum chamber, a wafer can only be removed from it) was satisfied. Thus there were still some errors in the translation of the requirements. This was later proven to be indeed the case for requirements one, four, five, six and ten.

Another problem, mainly during the first round of verification, consisted of syntax errors in the mcf files. For example, there were double ampersands (&&) in places where they simply were not allowed, a plus where no plus should have been and even mismatched brackets. Thanks to the PBES generator, all these errors were found and corrected.

During the process of rewriting some formulae, it turned out that there was one part of requirement five that was not really checking anything useful. Thus it was decided for this part to be removed from the requirement, which simply stated that eventually a wafer should be put into or taken out of the Low Vacuum chamber when there was a single wafer in the chamber.

During the validation of requirement one, we were unable to find an error in the mCRL2 model. To check whether or not the error was in the mCRL2 model or in the modal $\mu$-formulae, we manually checked the mCRL2 model using LTS Graph, the resulting graph is displayed in Figure 4. Using colors in the graph, it is clearly visible that the red Outer_Door(open) actions are only possible between the blue equalize(Sluice, Atm) actions and the green equalize(Sluice, LowVacuum) actions. Hence the mCRL2 model is correct and the modal $\mu$-formulae incorrect and need to be corrected.

Due to concerns about the translation of requirement nine to a modal $\mu$-formula, the model was checked by means of graphical representation. The large amount of states ask for the hiding of actions which are not important for the requirement and the removal of the second sluice. Each

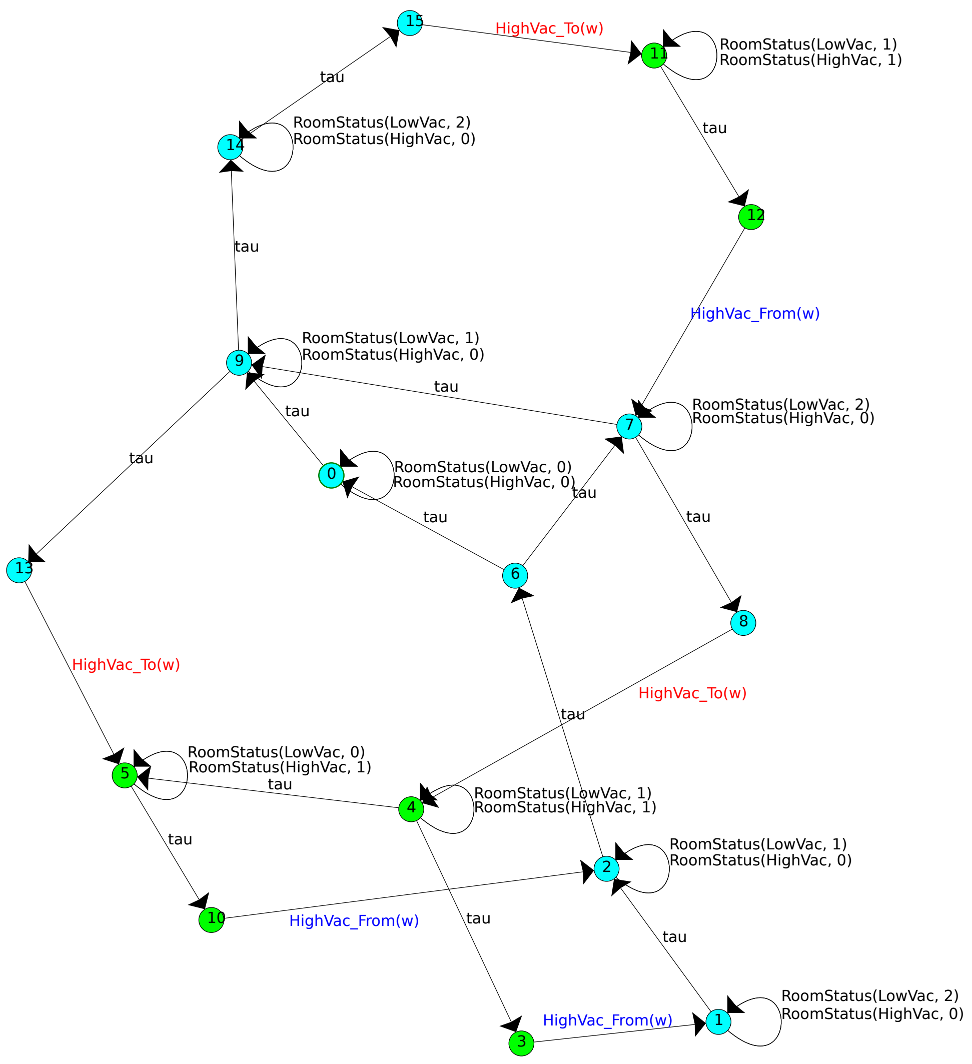


Figure 3: Reduced state space for Requirement four, part one.

sub-requirement is tested separately. To verify requirement nine, part one the graph depicted in Figure 5 is generated.

It is possible to derive the conclusion whether this requirement is fulfilled or not for this graph, but it takes a significant amount of time. To simplify the test, the model was adjusted slightly: for the requirements regarding the InnerDoor, the RoomFailure was disabled for OuterDoor actions by means of removing the OuterDoorState check from the model. This change does not influence the result of the testing of requirement nine, parts one and two. The graphical representation of this model is depicted in Figure 6. It can be seen that if the wrong InnerDoorState comes after the an InnerDoor_Open,

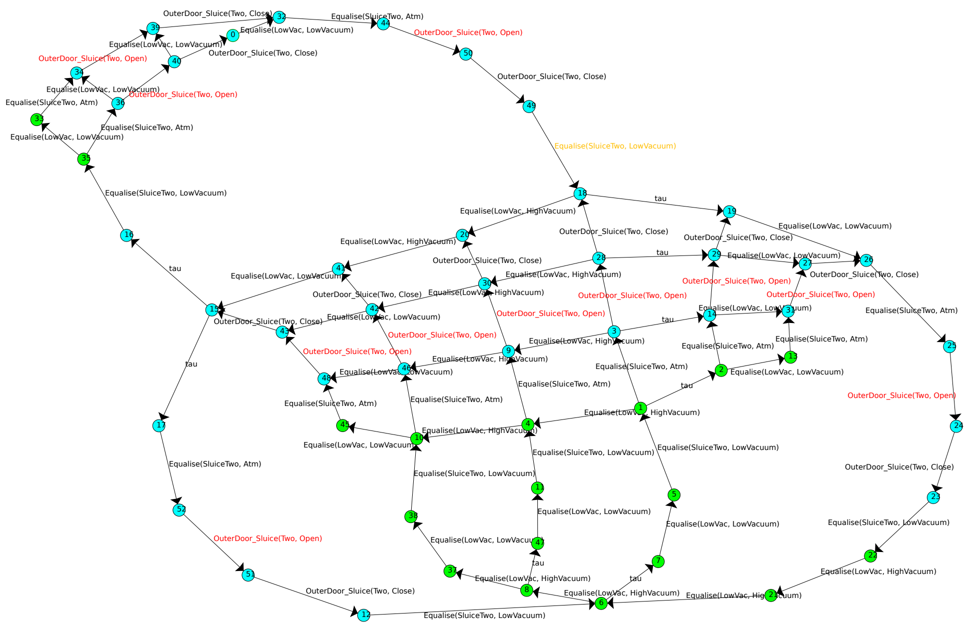


Figure 4: Reduced state space for requirement one.

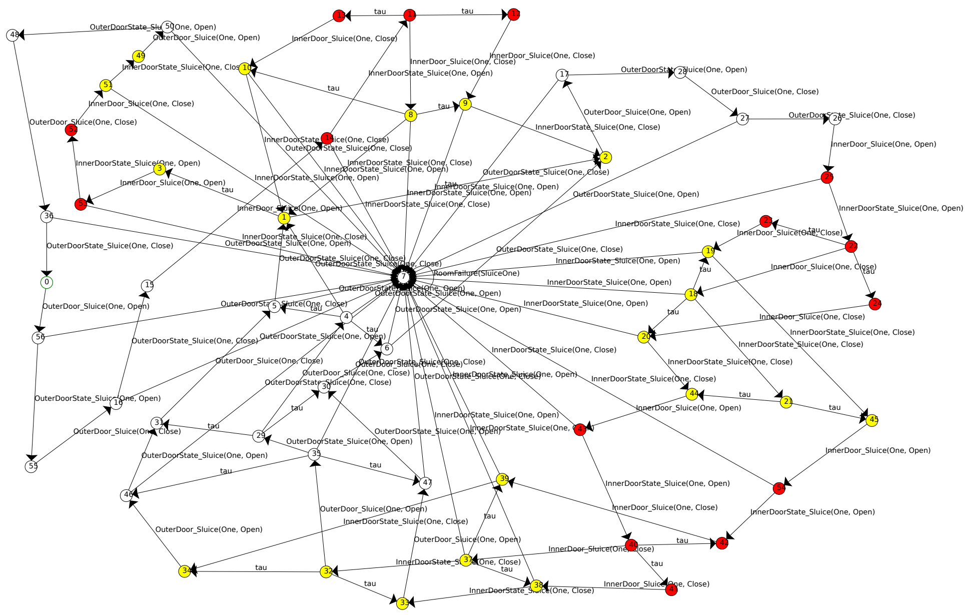
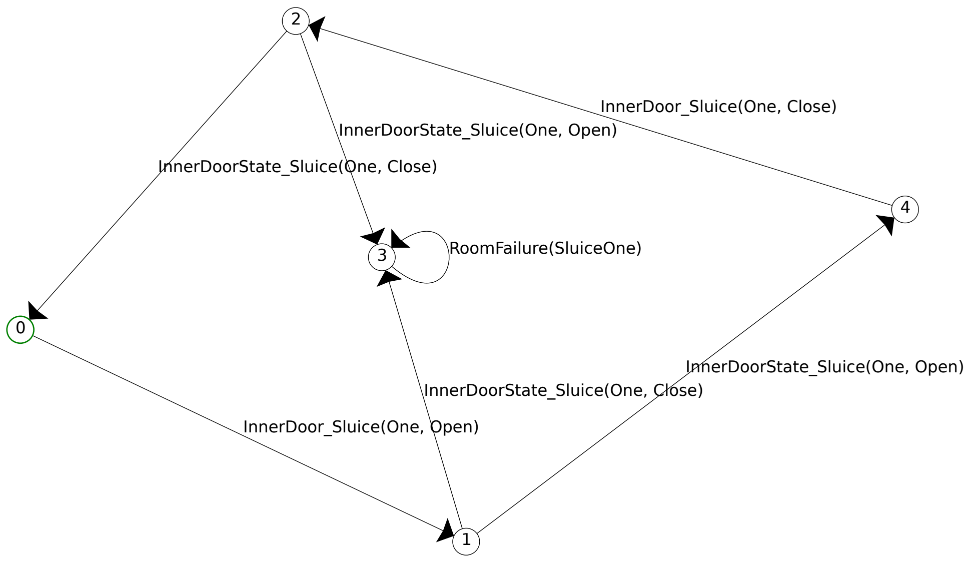


Figure 5: Reduced state space for requirement nine, parts one and two.

the system jumps to the failure state. Therefor, requirement nine, parts one and two are satisfied.



InnerDoor_Sluice(One, Close)
InnerDoorState_Sluice(One, Open)
InnerDoorState_Sluice(One, Close)
RoomFailure(SluiceOne)
InnerDoorState_Sluice(One, Open)
InnerDoorState_Sluice(One, Close)
InnerDoor_Sluice(One, Open)
0
1
2
3
4

Figure 6: Even further reduced state space for requirement nine, parts one and two.

To test requirement nine, parts three and four, the same simplification can be applied, but now for the OuterDoor. The resulting graph can be seen in Figure 7 and it can be seen that requirement nine, parts three and four are satisfied.

As with requirements nine, for requirement eight there were doubts about correctness of translation of the requirements into model $\mu$-formula. The requirements were checked with a graphical representation of the model, specially tailored for requirement eight. To simplify the graph only one sluice was used for this test and all actions except RoomFailure, HighVacDoor and HighVacDoorState were hidden. RoomFailure was temporary removed from sluice model as well. This has reduced graphical representation to only 25 states and 40 transitions. Figure 8 shows this representation. States where the door between the chambers is closed are marked in red, and states marked in yellow represent states where the door is open. It can be seen that after each HighVacDoorState, the next state will be RoomFailure if the data parameter of this action is not coherent with data parameter of the previously executed HighVacDoor action.

## 7.2 Errors due to unimplemented actions

The first version of the controller was a simple model. Hence, not all the actions were taken into account. This caused some of the requirements to appear as met, while in reality not all required actions were implemented. This was the case for requirements one, four, five, seven, eight, nine and ten.

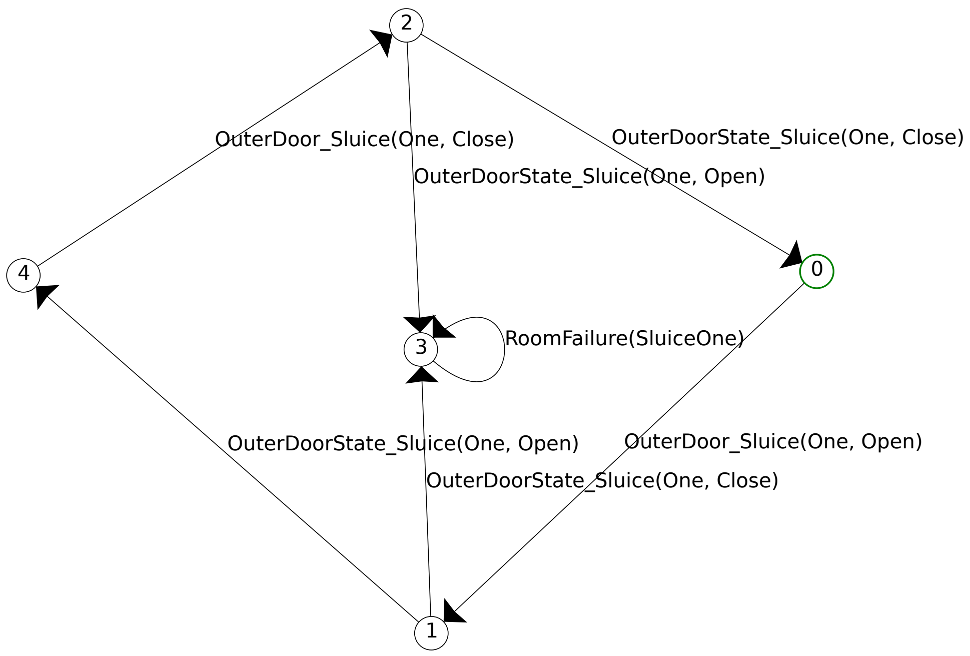

2
OuterDoor_Sluice(One, Close)
OuterDoorState_Sluice(One, Close)
OuterDoorState_Sluice(One, Open)
4
0
3
RoomFailure(SluiceOne)
OuterDoorState_Sluice(One, Open)
OuterDoor_Sluice(One, Open)
OuterDoorState_Sluice(One, Close)
1

Figure 7: Reduced state space for requirement nine, parts three and four.

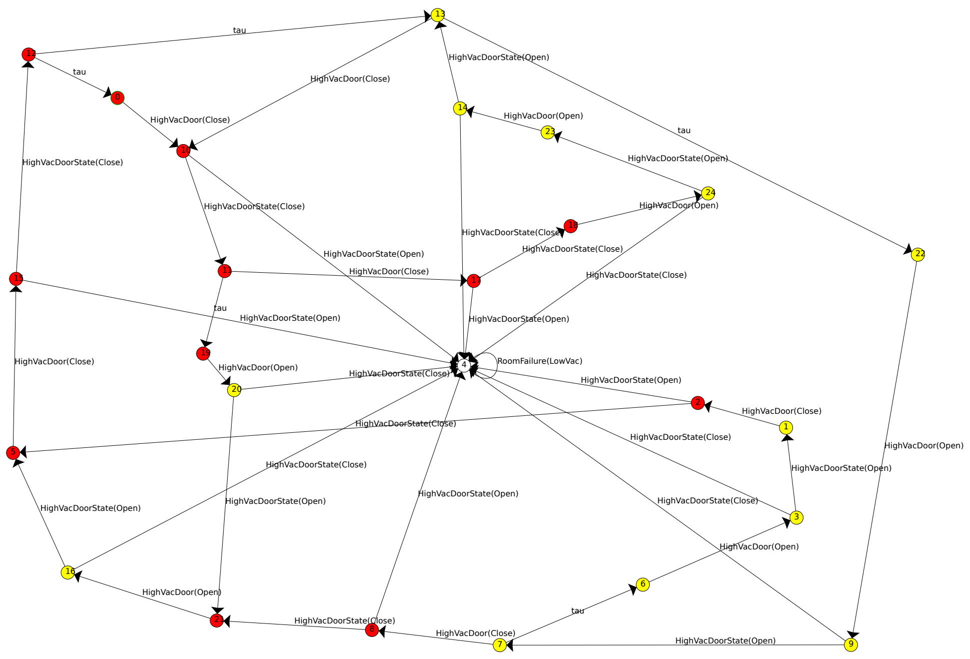


Figure 8: Reduced state space for requirement eight.

Fixing requirement seven needed two steps. First of all, we had to relax the requirement from “at all times or in every state it should be possible to request the status and get a response” to “after requesting the status eventually the system should respond”. This is necessary because it is not possible to add a status action to every possible state. Doing this would result in a state explosion. The second step to implement this requirement resulted in the addition of the Status and ReportStatus actions to the LowVacuum controller. To make sure that the ReportStatus action is executed with the correct number of wafers in the system, an addition had to be done to add up the wafers in the HighVacuum chamber to the wafers in the LowVacuum chamber. As wafers are only moved through the sluice on command of the LowVacuum controller, and no report status can be executed during that time, no wafers can be in the sluices when Status is executed. The addition is done by first converting the number of wafers in the high vacuum and low vacuum chambers into natural numbers, using the conversion functions bool2nat and stat2nat and adding the results using the built-in addition operator.

## 7.3 Errors due to a wrong model

Other errors were due to actual errors in the model. For requirement one, an action to measure the pressure in a room was required. After several unsuccessful attempts to implement this action, the requirement was relaxed. The controller now requires a sluice to equalize before opening a door, even if the pressure in the sluice is already at the required level. This introduces an extra load to the wafer stepper, but at least the vacuum inside the machine is not compromised this way.

Requirement three, part two states that the HighVacDoor can only be opened if all the InnerDoors are closed. In the original model, the Sluice and LowVac controllers synchronized on the transfer of a wafer. Thus it could happen that, after a wafer was transferred, the LowVac controller decided to transfer the wafer to the High Vacuum chamber before the InnerDoor was even closed. This obviously wrong behaviour was fixed by introducing an additional action, SluiceSync, which synchronizes the controllers after the Sluice controller has closed the InnerDoor.

## 7.4 Results

Table 4 shows the result of the verification of the current model. Nine requirements are met and one is not.

Table 4: The result of the verification and refinement process

| Requirement | Status |
|---|---|
| 1 | Fully satisfied |
| 2 | Fully satisfied |
| 3 | Fully satisfied |
| 4 | Fully satisfied |
| 5 | Fully satisfied |
| 6 | Fully satisfied |
| 7 | Fully satisfied |
| 8 | Fully satisfied |
| 9 | Fully satisfied |
| 10 | Not satisfied |

## 8 Conclusion

The result of this project is a controller for the ASML wafer stepper which meets nine out of ten of the requirements on the controller. Unfortunately, we were unable to refine the model in such a way that all ten requirements would be satisfied.

During the project, we have gained experience in developing a system based on a verifiable model. We learned how to apply the cycle of design, verification and refinement on a model-based design using mCRL2.

During the project, most out problems were due to the translation of our original requirements into Modal $\mu$-formulae.