Methodologies for Discrete Event Modelling and Simulation

Carleton University

March 3, 2025

Belal Alobieda 101151327

GarageDoor Project

Introduction

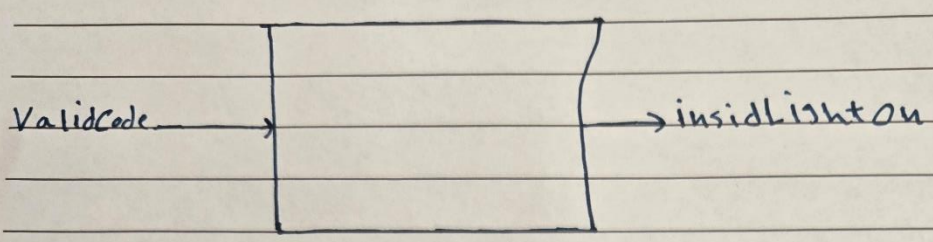
The purpose of this project is to show an understanding of modeling events using the DEVS formalism and to convert a previous simulation to the Cadmium V2 simulator. This project simulates a garage door control system that handles access codes, alarms, and lights. The system checks if a code is correct, decides whether to open the door, and triggers alarms or lights when needed. It helps demonstrate how different parts of the system work together and respond to events over time.

DEVS Models

Atomic Models

LightIn Model: Controls the indoor lighting system based on valid access codes. The model has one input and one output. The input, validCode, receives a signal when a correct access code is entered. The output, insideLightOn, controls whether the indoor light is turned on or off. When the model receives a valid code, it turns on the light and keeps it on for a set period. If another valid code is entered during this time, the light stays on longer. The model starts in the OFF state and switches to the ON state when a valid code is received. After the specified delay, it turns off unless another code is entered, ensuring the light remains active only when needed.

Atomic lightIn



lightIn = <X, Y, S, δint, δext, λ, ta>

X = {validCode}

Y = {insidLightOn}

[illegible]

S = {phase, σ, isLightOn, isSecondCode}

δint(S2) = S3, δint(S3) = S1, δint(S4) = S3

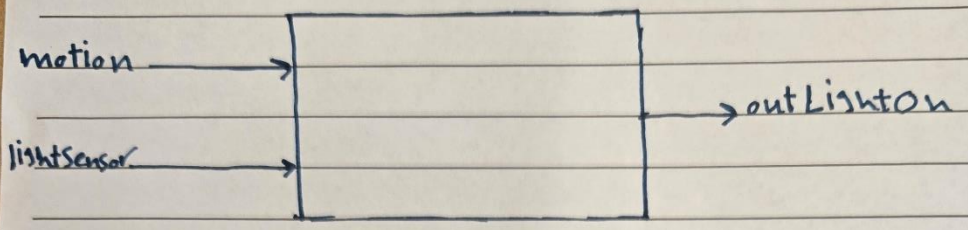
δext(S1, validCode) = S2

δext(S3, validCode) = S4

λ(S2 → S3) = insidLightOn = 1, λ(S3 → S1) = insidLightOn = 0

LightOut Model: Controls the outdoor lighting system based on motion detection and ambient light levels. The model has two inputs and one output. The motion input detects movement, while the lightSensor input determines whether it is daytime or nighttime. The output, outLightOn, controls whether the outdoor light is turned on. When motion is detected at night, the light turns on and stays on for a set period. If additional motion is detected within this time, the light remains on longer. The model starts in the OFF state and switches to the ON state when motion is detected in darkness. After the designated time, it turns off unless new motion is detected, ensuring the light is only active when necessary.

Atomic lightOut



lightout = <S, X, Y, δ_{int}, δ_{ext}, λ, ta>

X = {motion, lightSensor}

Y = {outLightOn}

S = {Phase, σ, isDayTime, isSecondTime}

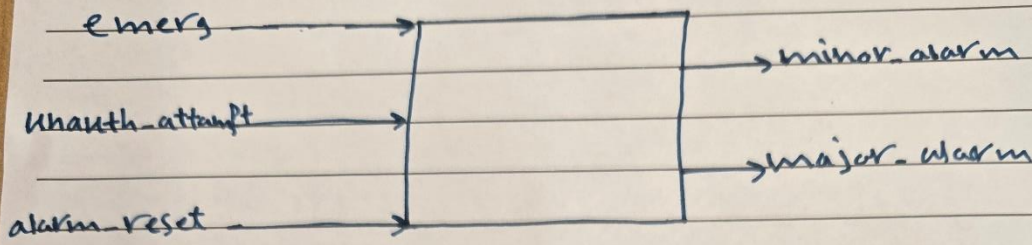
$\delta_{int}(s_2) = s_7$, $\delta_{int}(s_8) = s_1$, $\delta_{int}(s_6) = s_1$, $\delta_{int}(s_5) = s_4$, $\delta_{int}(s_3) = s_1$, $\delta_{int}(s_4) = s_1$

$\delta_{ext}(s_1, lightSensor=1) = s_2$, $\delta_{ext}(s_7, lightSensor=0) = s_8$, $\delta_{ext}(s_1, motion) = s_3$, $\delta_{ext}(s_4, lightSensor=1) = s_6$, $\delta_{ext}(s_4, motion) = s_5$

$\lambda(s_6 \to s_1)$ = outLightOn = 0, $\lambda(s_4 \to s_1)$ = outLightOn = 0, $\lambda(s_3 \to s_4)$ = outLightOn = 1

AlarmGen Model: Monitors and responds to unauthorized access attempts and emergency situations. The model has three inputs and two outputs. The unauth_attempt input detects unauthorized entry attempts, emerg signals emergency situations, and alarm_reset clears active alarms. The outputs, minor_alarm and major_alarm, activate depending on the severity of the detected event. When an unauthorized attempt occurs, a minor alarm is triggered, while an emergency event triggers a major alarm. If the reset signal is received, all alarms are turned off. The model starts in the IDLE state and moves to the ACTIVE state when an alarm is triggered. It remains active until reset, ensuring that security alerts are handled appropriately.

Atomic AlarGenerator



alarmGenerator = $\langle S, X, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$

$X = \{alarm\_reset, emerg, unauth\_attampt\}$

$Y = \{minor\_alarm, major\_alarm\}$

$S = \{phase, ta(\sigma), ifminor, ifmajor, ifreset\}$

$\delta_{int}(S_2) = S_1$, $\delta_{int}(S_3) = S_1$, $\delta_{int}(S_4) = S_1$

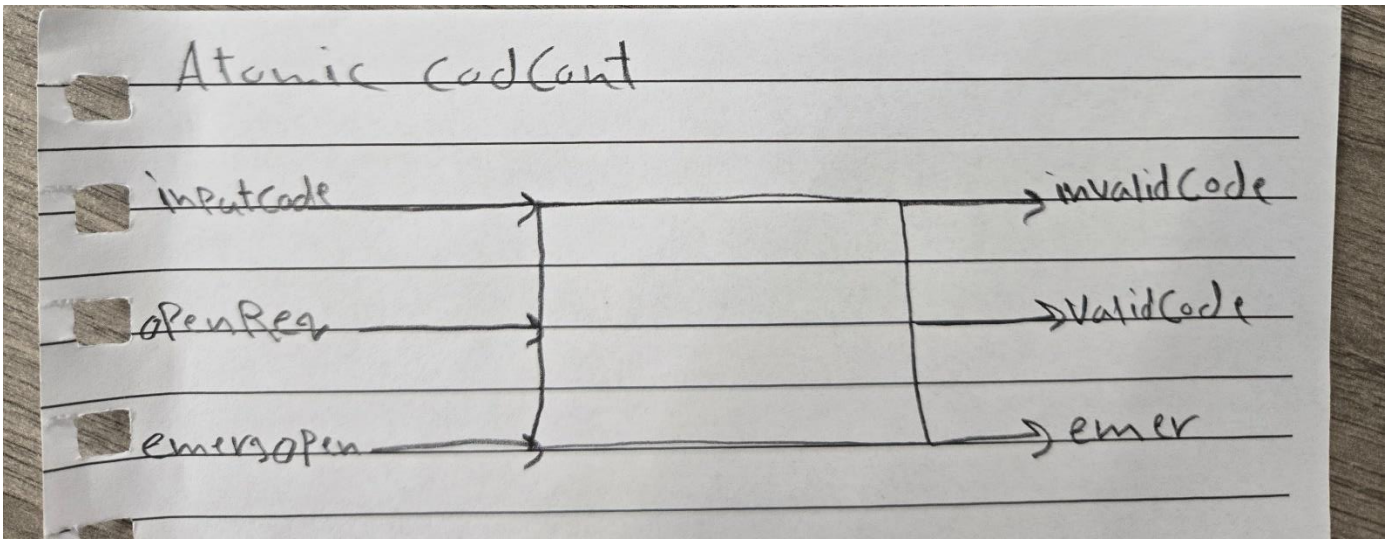
$\delta_{ext}(S_1, unauth\_attampt) = S_2$, $\delta_{ext}(S_1, alarm\_reset) = S_3$,

$\delta_{ext}(S_1, emerg) = S_4$

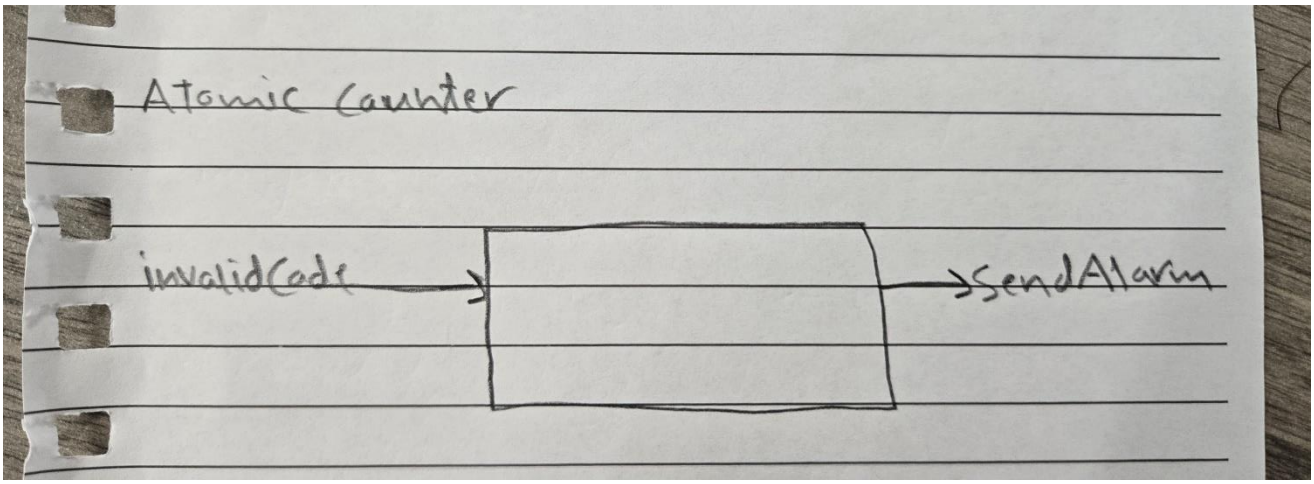
$\lambda(S_2 \to S_1) = minor\_alarm = 1$, $\lambda(S_3 \to S_1) = minor\_alarm = 0$,

$\lambda(S_4 \to S_1) = major\_alarm = 1$

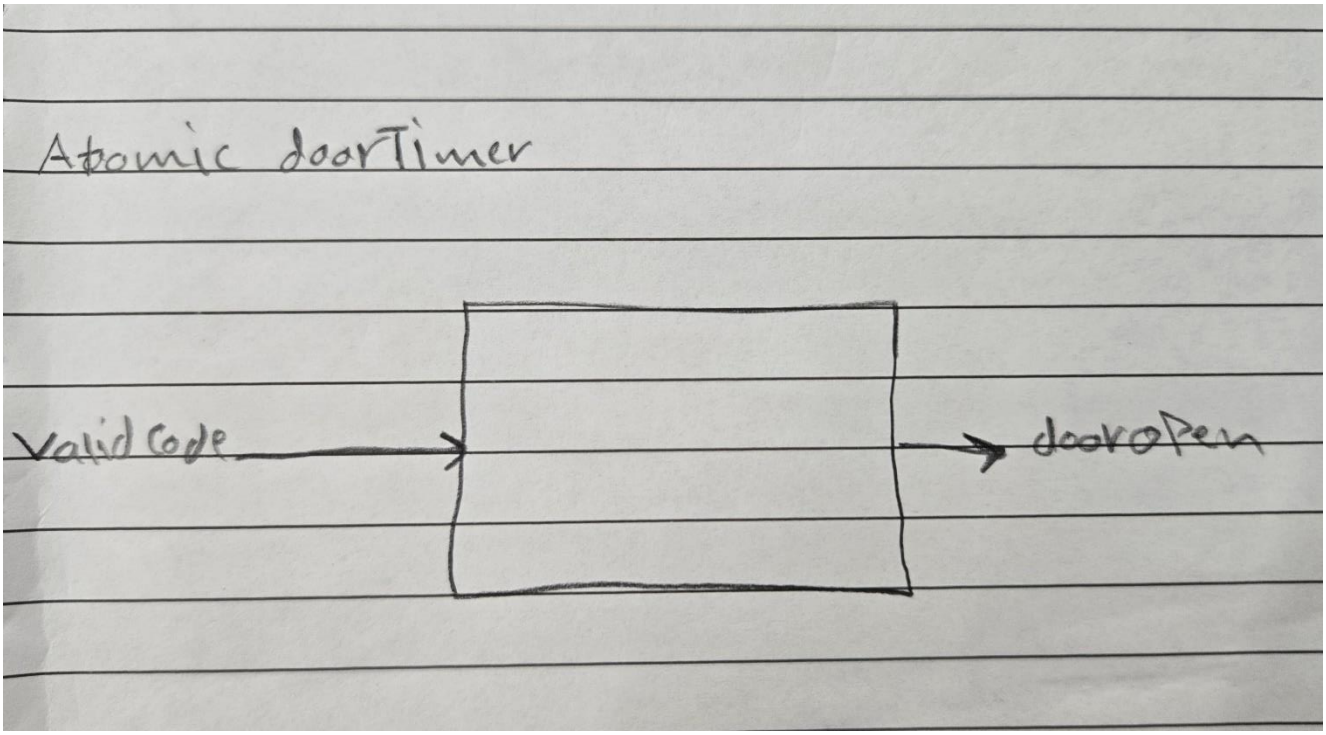
codCont Model: Handles access control by verifying user-entered codes and responding to emergency situations. The model has three inputs and three outputs. The inputCode input receives user-entered codes, openReq checks for door open requests, and emergOpen detects emergency open signals. The outputs include validCode to indicate a correct entry, invalidCode for incorrect attempts, and emer to signal an emergency situation. If a user enters the correct code, the door is authorized to open. If the code is incorrect, an invalid signal is sent. In emergencies, the model prioritizes immediate access. The system starts in the IDLE state and moves to the ACTIVE state when processing a code or emergency request. It resets back to IDLE after handling each event, ensuring proper security and controlled access



Counter Model: Monitors failed access attempts and triggers an alarm if multiple incorrect codes are entered within a set time. The model has one input and one output. The invalidCode input detects failed entry attempts, while the sendAlarm output activates an alarm when the failure limit is reached. If a user enters an incorrect code, the counter increases. If three incorrect attempts occur within a specified time (120 seconds), the model enters the SEND_ALARM state and triggers an alert. Otherwise, if too much time passes between attempts, the counter resets. The model starts in the IDLE state and transitions to SEND_ALARM when necessary, ensuring security by preventing repeated unauthorized access attempts

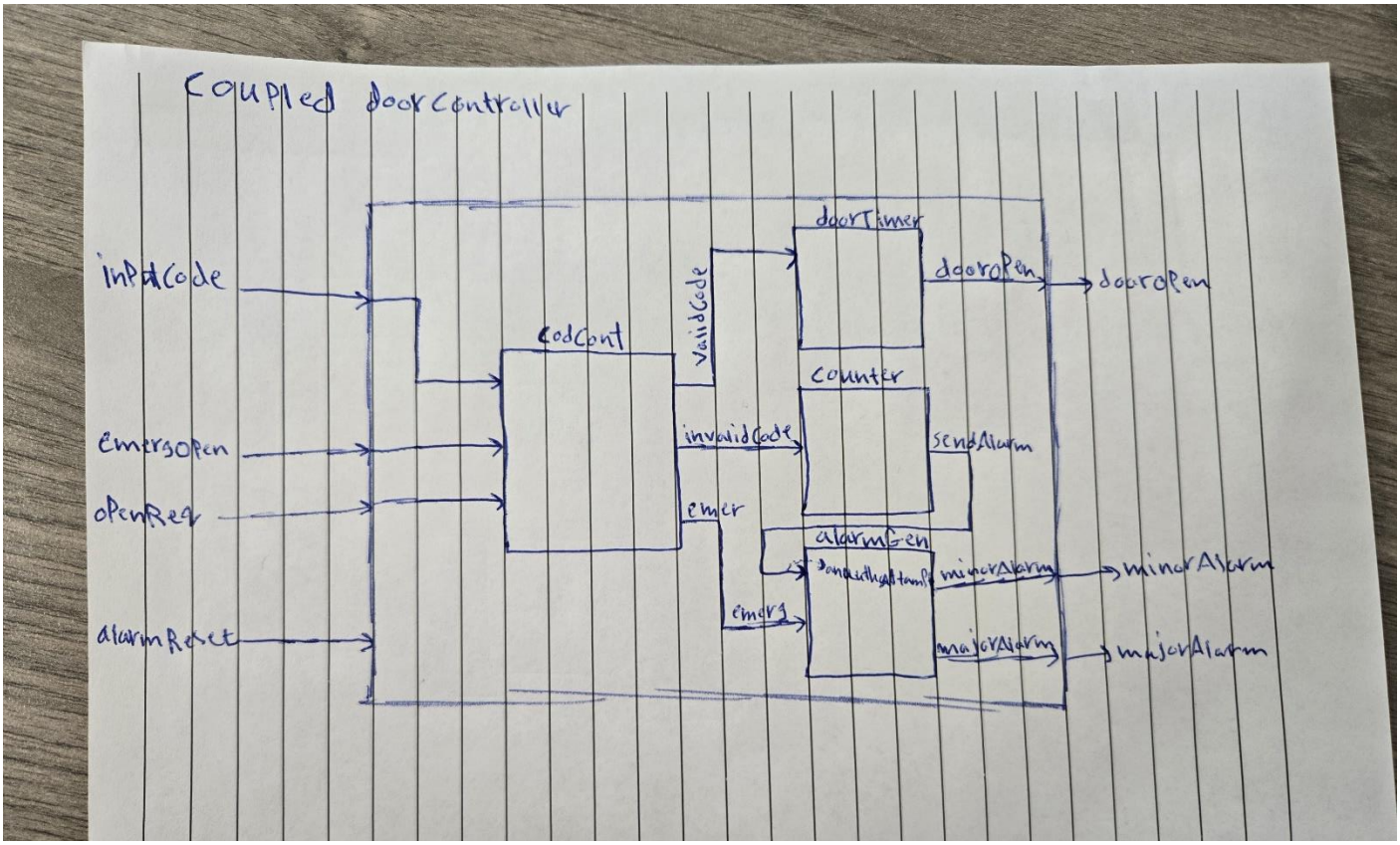


DoorTimer Model: Manages the automatic closing of the door after a valid entry. The model has one input and one output. The validCode input signals when an authorized access code has been entered, allowing the door to open. The doorOpen output indicates when the door is in the open state. When a valid code is received, the door opens and remains open for 15 seconds. If another valid code is entered within this time, the timer resets to extend the open period. Once the timer expires, the door automatically closes. The model starts in the CLOSE state and transitions to OPEN when access is granted, ensuring the door remains open only as needed for security and efficiency.



Coupled Models

DoorController Model: Manages the interactions between the access control, alarm, and door timer systems. The model has four inputs and three outputs. The inputCode receives user-entered codes, emergOpen detects emergency open signals, openReq processes door open requests, and alarmReset resets the alarm system. The outputs include doorOpen to indicate when the door is unlocked, minorAlarmOut for unauthorized access attempts, and majorAlarmOut for emergency situations. The model integrates multiple submodels, including codCont for code verification, counter for tracking failed attempts, doorTimer for automatic door closing, and AlarmGen for triggering alarms. It coordinates these components to ensure secure and controlled door access, prioritizing emergency situations while preventing unauthorized entry



DoorController = ⟨X, Y, D, {Mi}, EIC, EOC, IC, Select⟩

- **X (Input Set):**
  XXX = { inputCode: int, emergOpen: bool, openReq: bool, alarmReset: bool }
- **Y (Output Set):**
  Y = { doorOpen: bool, minorAlarmOut: bool, majorAlarmOut: bool }
- **D (Component Set):**
  D = { codCont, counter, doorTimer, alarmGen }
- **{Mi} (Submodels Mapping):**

  D(codCont)= M(codCont)

  D(counter)= M(counter)

  D(doorTimer)= M(doorTimer)

  D(alarmGen)= M(AlarmGen)
- **EIC (External Input Coupling):**

  (DoorController → inputCode, codCont → inputCode)

  (DoorController → emergOpen, codCont → emergOpen)

  (DoorController → openReq, codCont → openReq)

  (DoorController → alarmReset, alarmGen → alarm_reset)
- **EOC (External Output Coupling):**

  (doorTimer → doorOpen, DoorController → doorOpen)

  (alarmGen → minor_alarm, DoorController → minorAlarmOut)

  (alarmGen → major_alarm, DoorController → majorAlarmOut)
- **IC (Internal Coupling):**

  (codCont → validCode, doorTimer → validCode)

  (codCont → invalidCode, counter → invalidCode)

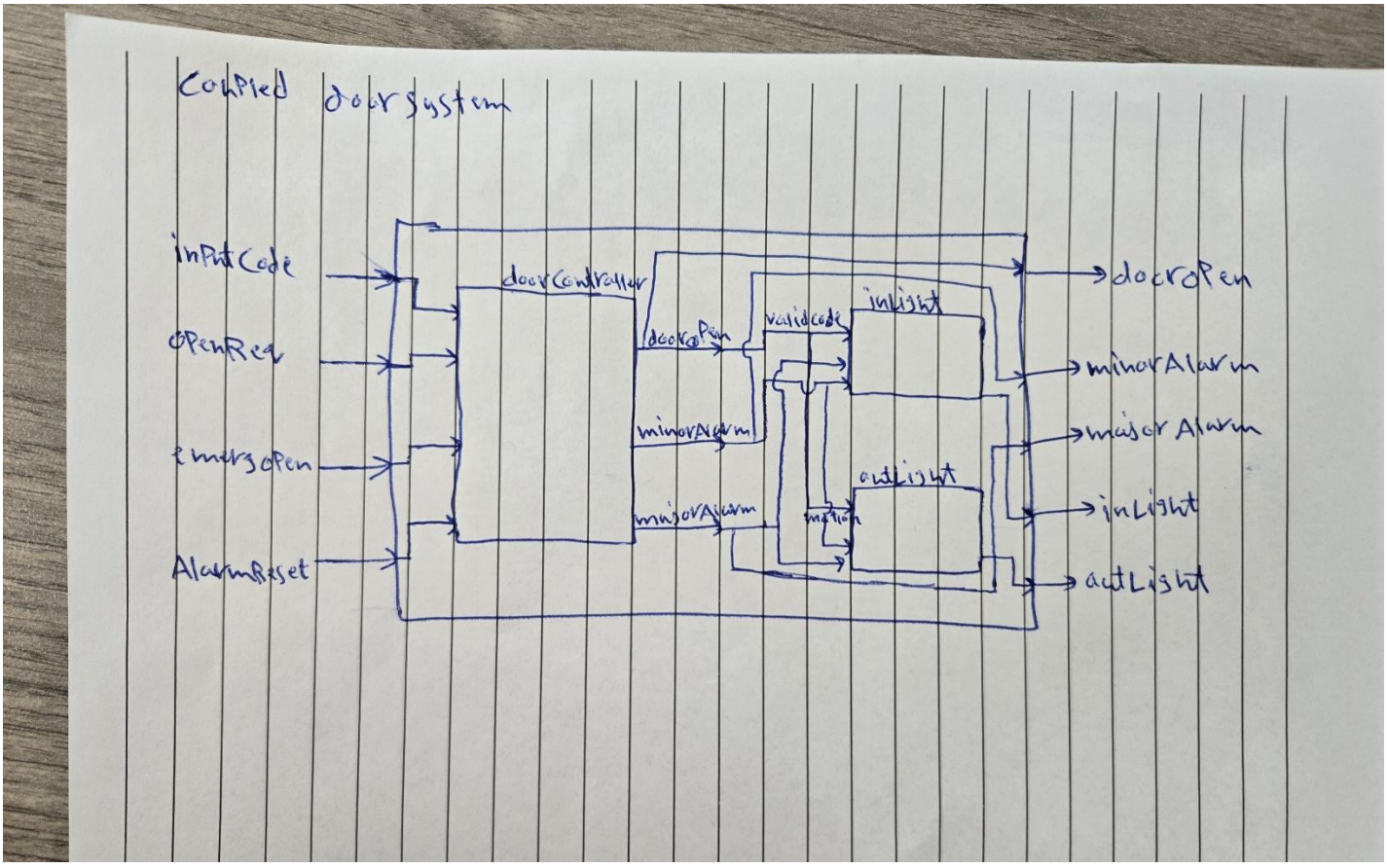
  (codCont → emer, alarmGen → emerg)

  (counter → sendAlarm, alarmGen → unauth_attempt)
- **Select Function:**

  {codCont, counter, doorTimer, alarmGen} = alarmGen (if an emergency occurs)

{codCont, counter, doorTimer} = codCont

The doorSystem model is a coupled DEVS model that integrates the door access control system with the lighting system, consisting of three submodels: doorController, which manages access control, alarms, and door operations; lightIn, which controls the indoor light based on door access; and lightOut, which manages outdoor lighting based on motion and alarm signals. The system receives external inputs such as access codes, emergency signals, and open requests, processing them to determine when to unlock the door and control the lights. Internal couplings allow doorController to send signals to lightIn and lightOut based on alarm conditions and motion detection, ensuring coordinated functionality



doorSystem = ⟨X, Y, D, {Mi}, EIC, EOC, IC, Select⟩

- **X (Input Set):**
  X = { outInputCode: int, outEmergOpen: bool, outOpenReq: bool, outAlarmReset: bool }
- **Y (Output Set):**
  Y = Ø (No external outputs directly from doorSystem)
- **D (Component Set):**
  D = { doorController, lightIn, lightOut }
- **{Mi} (Submodels Mapping):**

  D(doorController)= M(DoorController)

  D(lightIn)= M(LightIn)

  D(lightOut)= M(LightOut)D
- **EIC (External Input Coupling):**

  (doorSystem → outInputCode, doorController → inputCode)

  (doorSystem → outEmergOpen, doorController → emergOpen)

  (doorSystem → outOpenReq, doorController → openReq)

  (doorSystem → outAlarmReset, doorController → alarmReset)
- **EOC (External Output Coupling):**

  No external outputs are explicitly defined in this model.
- **IC (Internal Coupling):**

  (doorController → doorOpen, lightOut → motion)

  (doorController → minorAlarmOut, lightOut → lightSensor)

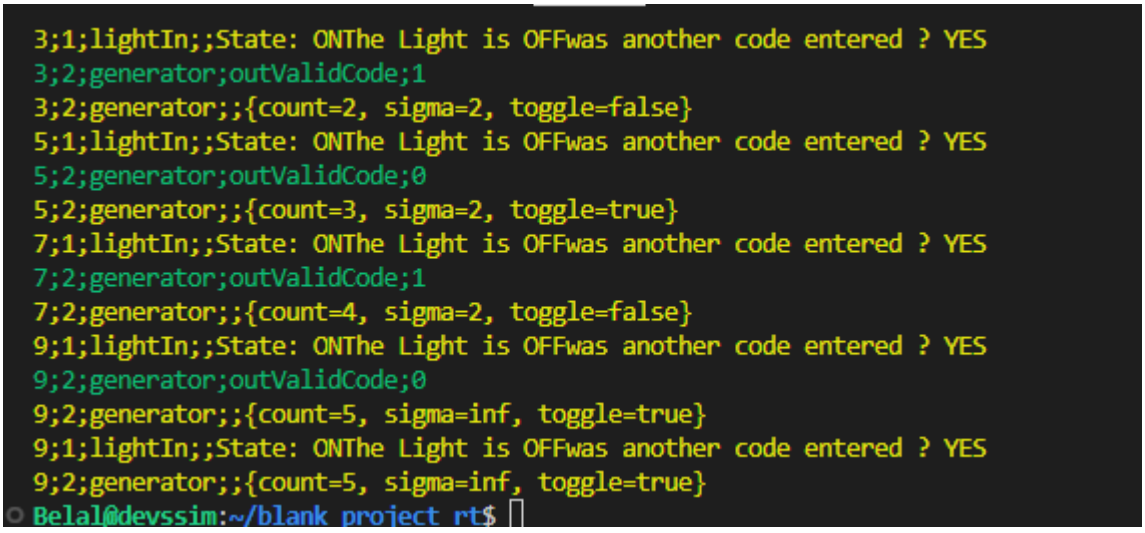
  (doorController → majorAlarmOut, lightIn → validCode)
- **Select Function:**

  {doorController, lightIn, lightOut} = doorController

Simulation Results

Atomic Model Testing

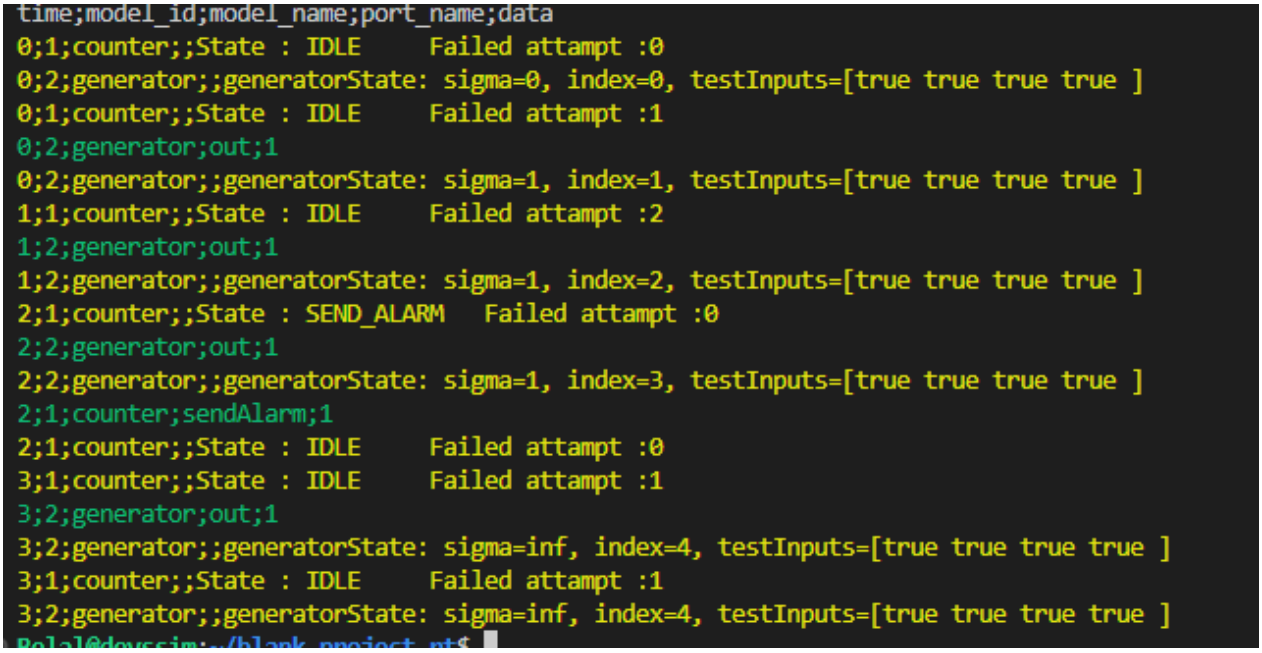
LightIn: To test the LightIn model, a generator will send a sequence of valid access codes. When the first code is received, the light should turn on for a set time. If another code arrives before the timer ends, the light stays on longer. This test ensures the model correctly handles state changes and timing updates. we can see that the LightIn model works as expected. When a valid code is received, the light turns on for 300 seconds (ta=300). If another code is entered during this time, the timer resets, keeping the light on longer.



lightOut test

```
Compilation done. Executable in the bin folder
Belal@devssim:~/blank_project_rt$ ./bin/sample_project
time;model_id;model_name;port_name;data
0;1;lightOut;;State: OFFit is a NightTimeanother code wasNOT ENTEREDSigma: inf
0;2;generator;;{count=0, sigma=2, motion=false, lightSensor=false}
2;1;lightOut;;State: ONit is a NightTimeanother code wasNOT ENTEREDSigma: 120
2;2;generator;outMotion;0
2;2;generator;outLightSensor;0
2;2;generator;;{count=1, sigma=2, motion=true, lightSensor=true}
4;1;lightOut;;State: ONit is a DayTimeanother code wasNOT ENTEREDSigma: 118
4;2;generator;outMotion;1
4;2;generator;outLightSensor;1
4;2;generator;;{count=2, sigma=2, motion=false, lightSensor=false}
6;1;lightOut;;State: ONit is a NightTimeanother code wasENTEREDSigma: 116
6;2;generator;outMotion;0
6;2;generator;outLightSensor;0
6;2;generator;;{count=3, sigma=2, motion=true, lightSensor=true}
8;1;lightOut;;State: ONit is a DayTimeanother code wasENTEREDSigma: 114
8;2;generator;outMotion;1
8;2;generator;outLightSensor;1
8;2;generator;;{count=4, sigma=2, motion=false, lightSensor=false}
10;1;lightOut;;State: ONit is a NightTimeanother code wasENTEREDSigma: 112
10;2;generator;outMotion;0
10;2;generator;outLightSensor;0
10;2;generator;;{count=5, sigma=2, motion=true, lightSensor=true}
12;1;lightOut;;State: ONit is a DayTimeanother code wasENTEREDSigma: 110
```

Counter test : in this test I made a vector with hard coded 4 true input to the counter system. Connected the out port to the invaliCode port. So the system supposed to send an alarm every 3 input and that what is happening.



codCont: in this atomic model we will do different test to check how the system react based on the input received on different ports. The first test is sending {true, false, false, true} on the emergOpen port. This one should activate an alarm and open the door. In the screenshot below,

it shows how the system react correctly

```
time;model_id;model_name;port_name;data
0;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
0;2;generator;;generatorState: sigma=0, index=0, testInputs=[true false false true ]
0;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS PRESSED  sigma : 0
0;2;generator;out;1
0;2;generator;;generatorState: sigma=1, index=1, testInputs=[true false false true ]
0;1;codCont;emer;1
0;1;codCont;validCode;1
0;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
1;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
1;2;generator;out;0
1;2;generator;;generatorState: sigma=1, index=2, testInputs=[true false false true ]
1;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
2;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
2;2;generator;out;0
2;2;generator;;generatorState: sigma=1, index=3, testInputs=[true false false true ]
2;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS PRESSED  sigma : 0
3;2;generator;out;1
3;2;generator;;generatorState: sigma=inf, index=4, testInputs=[true false false true ]
3;1;codCont;emer;1
3;1;codCont;validCode;1
3;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;2;generator;;generatorState: sigma=inf, index=4, testInputs=[true false false true ]
Belal@devssim:~/blank_project_rt$
```

second test is to send the same vector to the openReq port. And it should send an output on the validCode port to react properly. It should print a statement to say the code entered is valid or not valid.

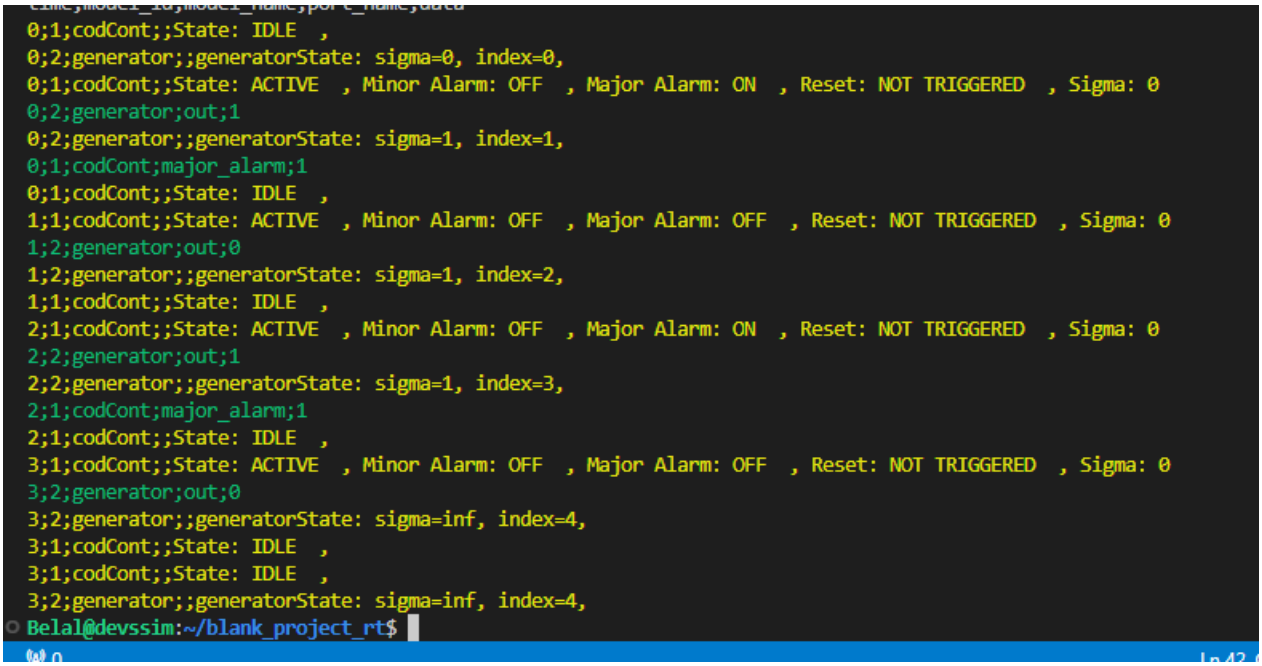
```
time;model_id;model_name;port_name;data
0;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
0;2;generator;;generatorState: sigma=0, index=0, testInputs=[true false false true ]
0;1;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
0;2;generator;out;1
0;2;generator;;generatorState: sigma=1, index=1, testInputs=[true false false true ]
0;1;codCont;validCode;1
0;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
1;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
1;2;generator;out;0
1;2;generator;;generatorState: sigma=1, index=2, testInputs=[true false false true ]
1;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
2;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
2;2;generator;out;0
2;2;generator;;generatorState: sigma=1, index=3, testInputs=[true false false true ]
2;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;1;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
3;2;generator;out;1
3;2;generator;;generatorState: sigma=inf, index=4, testInputs=[true false false true ]
3;1;codCont;validCode;1
3;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;1;codCont;;State :IDLE    the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : inf
3;2;generator;;generatorState: sigma=inf, index=4, testInputs=[true false false true ]
Belal@devssim:~/blank_project_rt$
```

Third test is entering a code and check if the system will be able to track the wrong ones. For that I made a vector that passes int to the port each time. The vector is {1234, 1234, 0000, 1212}. The system should only accept the valid code which is “1234”. Any other input, the system should print invalid code and send “true” to counter to keep track of how many invalid code received.

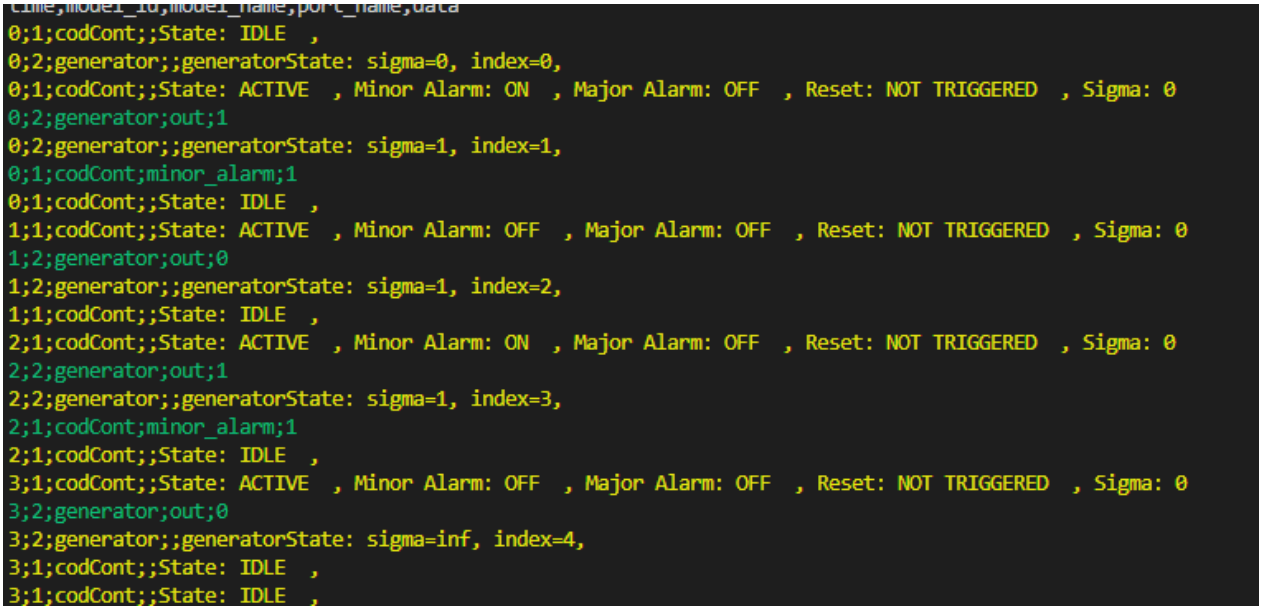
```
time;model_id;model_name;port_name;data
0;1;codCont;;State :IDLE
0;2;generator;;generatorState: sigma=0, index=0,
0;1;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
0;2;generator;out;1234
0;2;generator;;generatorState: sigma=1, index=1,
0;1;codCont;validCode;1
0;1;codCont;;State :IDLE
1;1;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
1;2;generator;out;1234
1;2;generator;;generatorState: sigma=1, index=2,
1;1;codCont;validCode;1
1;1;codCont;;State :IDLE
2;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
2;2;generator;out;0
2;2;generator;;generatorState: sigma=1, index=3,
2;1;codCont;invalidCode;1
2;1;codCont;;State :IDLE
3;1;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
3;2;generator;out;1212
3;2;generator;;generatorState: sigma=inf, index=4,
3;1;codCont;invalidCode;1
3;1;codCont;;State :IDLE
3;1;codCont;;State :IDLE
3;2;generator;;generatorState: sigma=inf, index=4,
```

alarmGenerator.

first test**:** in this atomic model we will also do more than one test to test each port. We will start by testing the emerg port which supposed to receive an input from the door controller with boolean if an emergency button was pressed. this should activate major alarm and open the door rightaway. In this screenshot I made sure that it’s easy to read and follow and deleted all the extra print statements. The vector of Boolean I am passing here is {true, false, true, false}



Second test: in this test we will try to trigger the minor alarm. This normally should come from counter after three 3 an authorized attempt. In my test I am passing same vector with {true, false, true, false}.



Third test: in this one we are testing if the alarm reset works. I made small modification to the code to make the system always send major and minor alarm. I did that to test what would

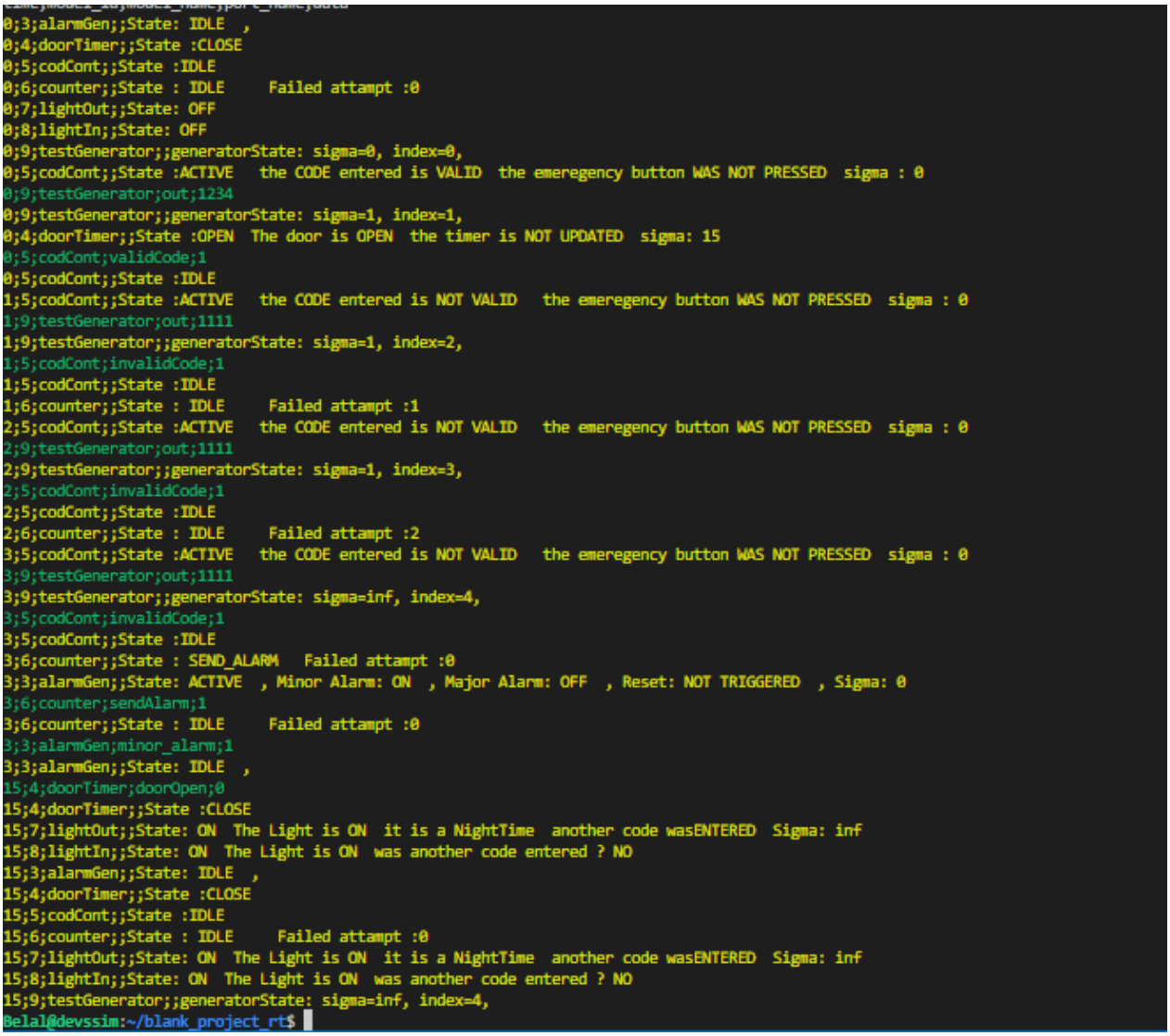
happen when I trigger the alarm reset. In the generator I used the same vector with {false, false, true, false}. This shouldn't reset the alarms until it reaches the true.

```
0;1;codCont;;State: IDLE  ,
0;2;generator;;generatorState: sigma=0, index=0,
0;1;codCont;;State: ACTIVE  , Minor Alarm: ON  , Major Alarm: ON  , Reset: NOT TRIGGERED  , Sigma: 0
0;2;generator;out;0
0;2;generator;;generatorState: sigma=1, index=1,
0;1;codCont;major_alarm;1
0;1;codCont;;State: IDLE  ,
1;1;codCont;;State: ACTIVE  , Minor Alarm: ON  , Major Alarm: ON  , Reset: NOT TRIGGERED  , Sigma: 0
1;2;generator;out;0
1;2;generator;;generatorState: sigma=1, index=2,
1;1;codCont;major_alarm;1
1;1;codCont;;State: IDLE  ,
2;1;codCont;;State: ACTIVE  , Minor Alarm: OFF  , Major Alarm: OFF  , Reset: TRIGGERED  , Sigma: 0
2;2;generator;out;1
2;2;generator;;generatorState: sigma=1, index=3,
2;1;codCont;minor_alarm;0
2;1;codCont;major_alarm;0
2;1;codCont;;State: IDLE  ,
3;1;codCont;;State: ACTIVE  , Minor Alarm: ON  , Major Alarm: ON  , Reset: NOT TRIGGERED  , Sigma: 0
3;2;generator;out;0
3;2;generator;;generatorState: sigma=inf, index=4,
3;1;codCont;major_alarm;1
3;1;codCont;;State: IDLE  ,
3;1;codCont;;State: IDLE  ,
3;2;generator;;generatorState: sigma=inf, index=4,
```

doorController coupled model: this coupled model contains four different atomic model. It has the counter, codCont, alarmGen, doorTimer. The coupled model was tested with the same vectore but different inputs were giving. I used {1234, 1111, 1111, 111}. This should give us the first input code to be correct and the other three to be wrong input code. This should start by opening the door of garage. And then after each failed trial, the system should keep count of that and send a minor alarm after the third one and reset the counter to 0 failed.

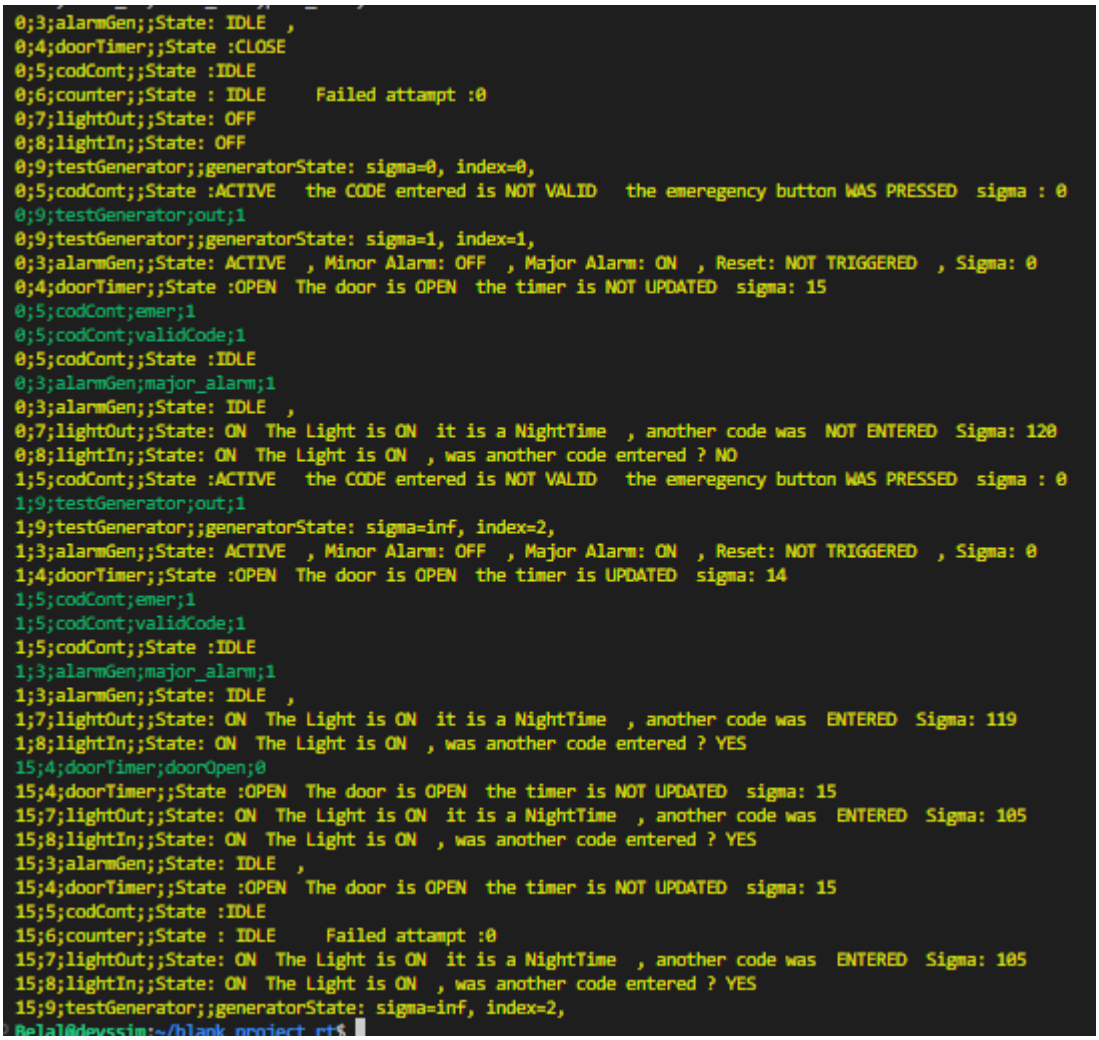
```
0;2;alarmGen;;State: IDLE  ,
0;3;doorTimer;;State :CLOSE
0;4;codCont;;State :IDLE
0;5;counter;;State : IDLE     Failed attampt :0
0;6;testGenerator;;generatorState: sigma=0, index=0,
0;4;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
0;6;testGenerator;out;1234
0;6;testGenerator;;generatorState: sigma=1, index=1,
0;3;doorTimer;;State :OPEN  The door is OPEN  the timer is NOT UPDATED  sigma: 15
0;4;codCont;validCode;1
0;4;codCont;;State :IDLE
1;4;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
1;6;testGenerator;out;1111
1;6;testGenerator;;generatorState: sigma=1, index=2,
1;4;codCont;invalidCode;1
1;4;codCont;;State :IDLE
1;5;counter;;State : IDLE     Failed attampt :1
2;4;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
2;6;testGenerator;out;1111
2;6;testGenerator;;generatorState: sigma=1, index=3,
2;4;codCont;invalidCode;1
2;4;codCont;;State :IDLE
2;5;counter;;State : IDLE     Failed attampt :2
3;4;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
3;6;testGenerator;out;1111
3;6;testGenerator;;generatorState: sigma=inf, index=4,
3;4;codCont;invalidCode;1
3;4;codCont;;State :IDLE
3;5;counter;;State : SEND_ALARM   Failed attampt :0
3;2;alarmGen;;State: ACTIVE  , Minor Alarm: ON  , Major Alarm: OFF  , Reset: NOT TRIGGERED  , Sigma: 0
3;5;counter;sendAlarm;1
3;5;counter;;State : IDLE     Failed attampt :0
3;2;alarmGen;minor_alarm;1
3;2;alarmGen;;State: IDLE  ,
15;3;doorTimer;doorOpen;0
```

doorSystemCoupled: this coupled system contains two atomic model and one coupled model. It contains the lightIn and lightOut atomic models. And it has the doorController coupled model. Testing the system with the same input to check if the light will turn on.



System Testing

Now after coupling everything together we are going have some tests on the whole system together. I will start with the emergency button being pushed. that should send an alarm to open the door and to turn the lights on if it’s night.



Second test: we will have vector carrying 5 inputs as follow {false, true, false, false, false}. The first one should be counted as an authorized attempt. The second should open the door. The fourth one should trigger a minor alarm and reset the counter for the failed attempt. And the last one should be counter again as one an authorized attempt.

```
0;3;alarmGen;;State: IDLE  ,
0;4;doorTimer;;State :CLOSE
0;5;codCont;;State :IDLE
0;6;counter;;State : IDLE     Failed attampt :0
0;7;lightOut;;State: OFF
0;8;lightIn;;State: OFF
0;9;testGenerator;;generatorState: sigma=0, index=0,
0;5;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
0;9;testGenerator;out;1212
0;9;testGenerator;;generatorState: sigma=1, index=1,
0;5;codCont;invalidCode;1
0;5;codCont;;State :IDLE
0;6;counter;;State : IDLE     Failed attampt :1
1;5;codCont;;State :ACTIVE   the CODE entered is VALID  the emeregency button WAS NOT PRESSED  sigma : 0
1;9;testGenerator;out;1234
1;9;testGenerator;;generatorState: sigma=1, index=2,
1;4;doorTimer;;State :OPEN  The door is OPEN  the timer is NOT UPDATED  sigma: 15
1;5;codCont;validCode;1
1;5;codCont;;State :IDLE
2;5;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
2;9;testGenerator;out;1235
2;9;testGenerator;;generatorState: sigma=1, index=3,
2;5;codCont;invalidCode;1
2;5;codCont;;State :IDLE
2;6;counter;;State : IDLE     Failed attampt :2
3;5;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
3;9;testGenerator;out;1111
3;9;testGenerator;;generatorState: sigma=1, index=4,
3;5;codCont;invalidCode;1
3;5;codCont;;State :IDLE
3;6;counter;;State : SEND_ALARM   Failed attampt :0
3;3;alarmGen;;State: ACTIVE  , Minor Alarm: ON  , Major Alarm: OFF  , Reset: NOT TRIGGERED  , Sigma: 0
3;6;counter;sendAlarm;1
3;6;counter;;State : IDLE     Failed attampt :0
3;3;alarmGen;minor_alarm;1
3;3;alarmGen;;State: IDLE  ,
4;5;codCont;;State :ACTIVE   the CODE entered is NOT VALID   the emeregency button WAS NOT PRESSED  sigma : 0
4;9;testGenerator;out;2222
4;9;testGenerator;;generatorState: sigma=inf, index=5,
4;5;codCont;invalidCode;1
4;5;codCont;;State :IDLE
4;6;counter;;State : IDLE     Failed attampt :1
16;4;doorTimer;doorOpen;0
16;4;doorTimer;;State :CLOSE
16;7;lightOut;;State: ON  The Light is ON  it is a NightTime  , another code was  NOT ENTERED  Sigma: 120
16;8;lightIn;;State: ON  The Light is ON  , was another code entered ? NO
16;3;alarmGen;;State: IDLE  ,
16;4;doorTimer;;State :CLOSE
16;5;codCont;;State :IDLE
16;6;counter;;State : IDLE     Failed attampt :1
16;7;lightOut;;State: ON  The Light is ON  it is a NightTime  , another code was  NOT ENTERED  Sigma: 120
16;8;lightIn;;State: ON  The Light is ON  , was another code entered ? NO
16;9;testGenerator;;generatorState: sigma=inf, index=5,
```

From this simulation we can see that everything we expected to happen happened. And the light was set on since it’s a night. The system works as it supposed to.