EEL3090 : Embedded System

Course Project B4

Deriving Finite state machine (FSM)
From Verilog descriptions in generalized manner

Yash Kumawat(B19EE091) , Yogesh Nema(B19EE092)

Objective : The objective of this project is to develop a **C++** model to extract a finite state machine from the Verilog description.

Finite state machine (FSM):

The basic idea of an FSM is to store a sequence of different unique states and transition between them depending on the values of the inputs and the current state of the machine.

Model Description :

We have developed a **C++** model which takes a Verilog file as an input and gives FSM as an output and checks the model on 5 different Verilog input files.
For designing this model we have mainly focused on 3 different conditions:

1. If-else
2. If-else if-else
3. Nested if else

Code Description :

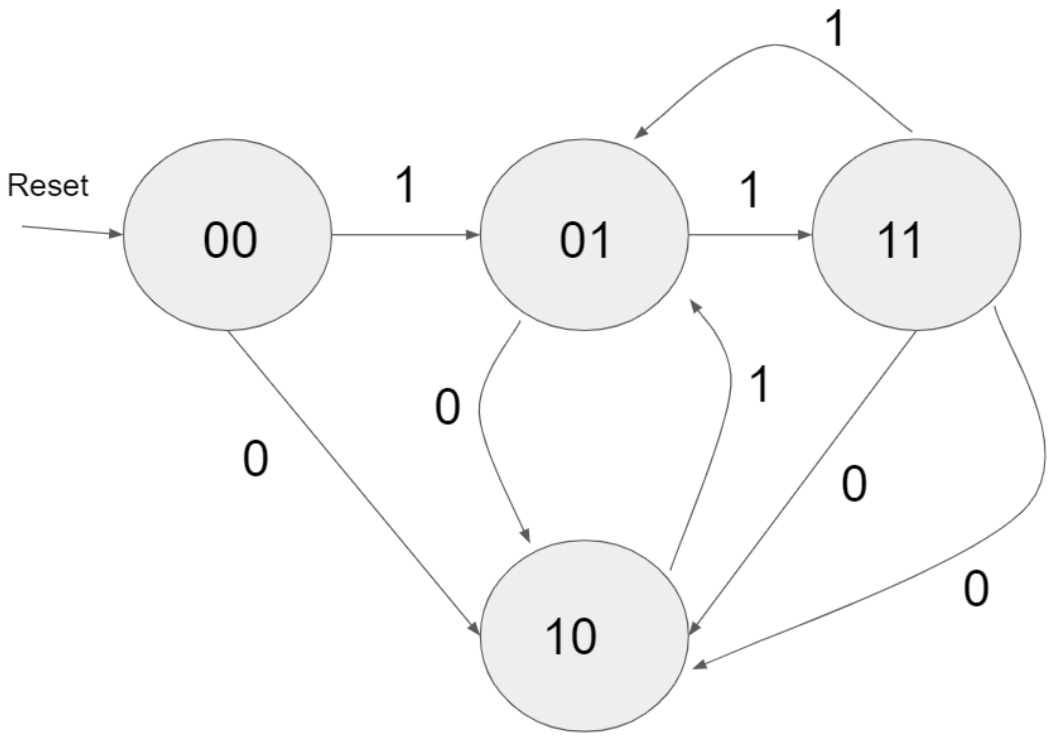
- First of all we convert the whole code into a 2-D vector of string. In one vector of string we store the words which are in a single line.
- Now first of all we detect the case block in the formed 2-D vector.We also ensure that this case block includes FSM as there are many case blocks present in code.
- After that we check when the case of a particular state is starting.
- After that if there is a condition for the changing of state and what is that condition by detecting if /else if /else condition.
- After that we detect the final state after the applied condition.
- We terminate our code when we detect an “endcase” string in the vector.

❖ FSM for 1st verilog input file - input1.v

Case block of the input1 file which is necessary for designing the FSM :

```
case (state)
2'b00:
begin
if (inp) state <= 2'b01;
else state <= 2'b10;
end
2'b01:
begin
if (inp) state <= 2'b11;
else state <= 2'b10;
end
2'b10:
begin
if (inp) state <= 2'b01 ;
else state <= 2'b11 ;
end
2'b11:
begin
if (inp) state <= 2'b01 ;
else state <= 2'b10 ;
end
endcase
```

FSM : Manually



FSM : By code

```
Enter The Name of The Verilog File : input1.v

/*********************************************************

        Finite State Machine For input1.v

*********************************************************/

2'b00 ------> if  (inp) == 1  ------> 2'b01;

2'b00 ------> else : ------> 2'b10;

2'b01 ------> if  (inp) == 1  ------> 2'b11;

2'b01 ------> else : ------> 2'b10;

2'b10 ------> if  (inp) == 1  ------> 2'b01

2'b10 ------> else : ------> 2'b11

2'b11 ------> if  (inp) == 1  ------> 2'b01

2'b11 ------> else : ------> 2'b10
```

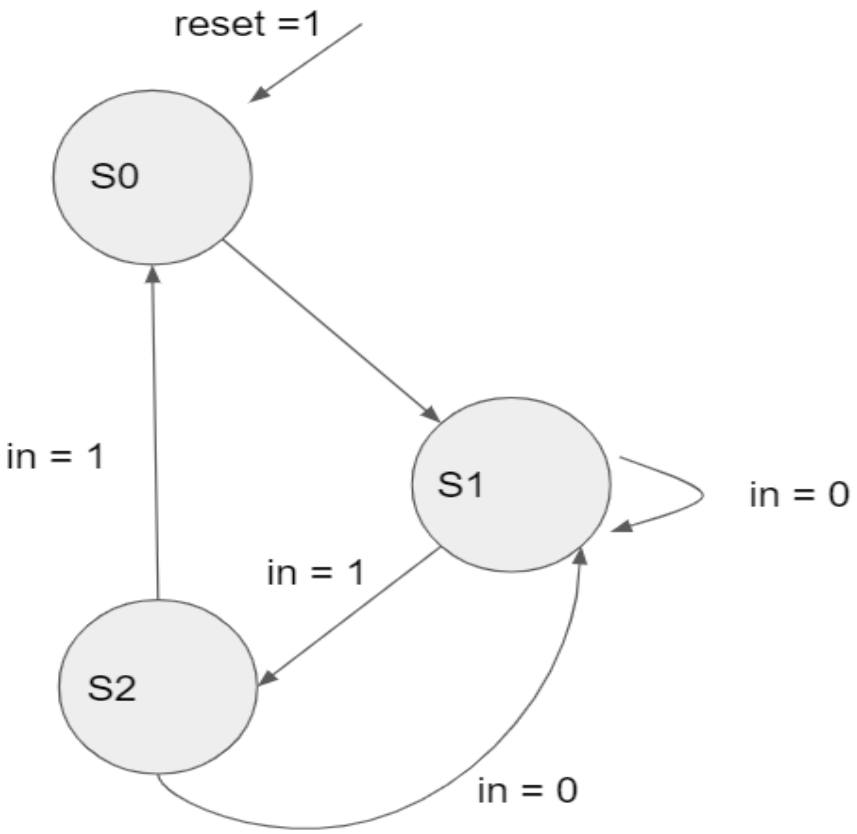
- **In this question states are 2'b00, 2'b01, 2'b10 , 2'b11.**
- **We have successfully extracted FSM from the code.**

❖ FSM for 2nd verilog input file - input2.v

Case block of the input2 file which is necessary for designing the FSM :

```
case (state)
S0:
state = S1 ;
S1:
if (in)
state = S2 ;
else
state = S1 ;
S2:
if (in)
state = S0 ;
else
state = S1 ;
endcase
```

FSM : Manually



FSM : By code

```
Enter The Name of The Verilog File : input2.v

/*****************************************************************

                              Finite State Machine For input2.v

*****************************************************************

S0 ----> S1

S1 ------> if  (in) == 1  ----> S2

S1 ------> else : ------> ----> S1

S2 ------> if  (in) == 1  ----> S0

S2 ------> else : ------> ----> S1
```

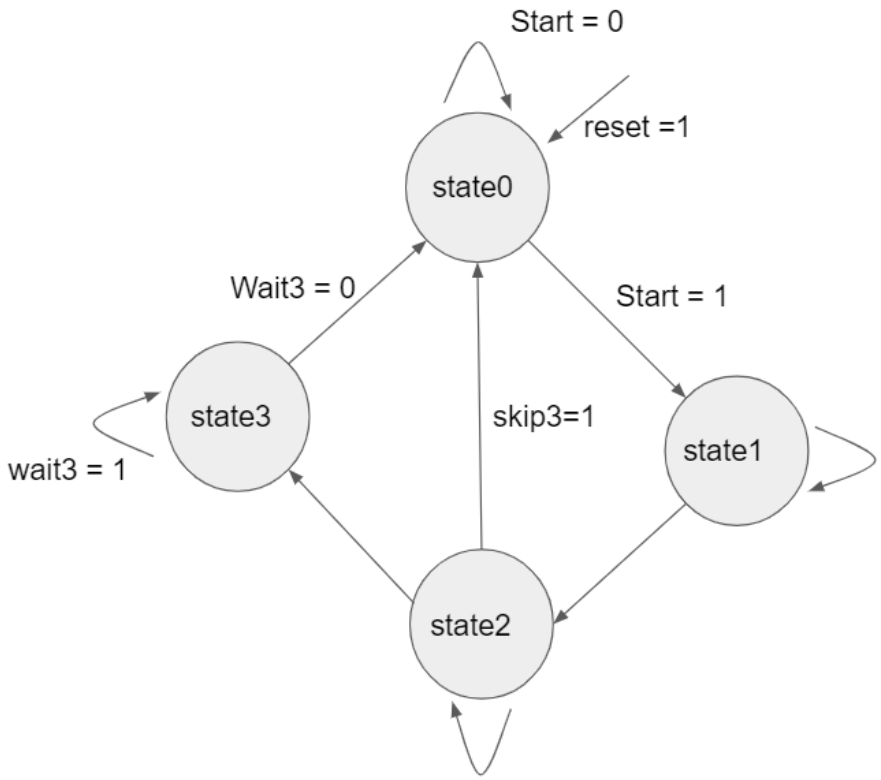
❖ FSM for 3rd verilog input file - input3.v

Case block of the input3 file which is necessary for designing the FSM :

```
case (state)
state0:
begin
if (start) nxt_st = state1 ;
else nxt_st = state0 ;
end
state1: begin
nxt_st = state2 ;
end
state2: begin
if (skip3) nxt_st = state0 ;
else nxt_st = state3 ;
end
state3: begin
if (wait3) nxt_st = state3 ;
else nxt_st = state0 ;
end
default: nxt_st = state0 ;
endcase
```

In the above verilog code we can see that “if” condition is defined in one line so for this case our model is working properly.

FSM : Manually



FSM : By code

```
Enter The Name of The Verilog File : input3.v

/*************************************************************************

                          Finite State Machine For input3.v

*************************************************************************
state0 ------> if  (start) == 1  ------> state1

state0 ------> else : ------> state0

state1 ----> state2

state2 ------> if  (skip3) == 1  ------> state0

state2 ------> else : ------> state3

state3 ------> if  (wait3) == 1  ------> state3

state3 ------> else : ------> state0
```

❖ FSM for 4th verilog input file - input4.v

Case block of the input4 file which is necessary for designing the FSM :

```
case (state)//synopsys full_case parallel_case
IDLE:
begin
pid_le_sm = 1'b1;
if (rx_valid && rx_active)
next_state = ACTIVE ;
end
ACTIVE:
begin
if (pid_ACK && !rx_err) begin
got_pid_ack = 1'b1;
next_state = IDLE ;
end
else if (pid_TOKEN && rx_valid && rx_active && !rx_err) begin
token_le_1 = 1'b1;
next_state = TOKEN ;
end
else if (pid_DATA && rx_valid && rx_active && !rx_err) begin
data_valid_d = 1'b1;
next_state = DATA ;
end
else if (!rx_active || rx_err || (rx_valid && !(pid_TOKEN || pid_DATA))) begin
next_state = IDLE ;
end
end
TOKEN:
begin
if (rx_valid && rx_active && !rx_err) begin
token_le_2 = 1'b1;
next_state = IDLE ;
end
else if (!rx_active || rx_err) begin
next_state = IDLE ;
end
end
DATA:
begin
if (rx_valid && rx_active && !rx_err)
data_valid_d = 1'b1;
next_state = IDLE ;
end
end
endcase
```

FSM : By code

```
/*****************************************************************************************************************

                            Finite State Machine For input4.v

*****************************************************************************************************************

IDLE ------> if  (rx_valid && rx_active) == 1  ----> ACTIVE

ACTIVE ------> if  (pid_ACK && !rx_err) == 1  ----> IDLE

ACTIVE ------> else  if (pid_TOKEN && rx_valid && rx_active && !rx_err) == 1  ----> TOKEN

ACTIVE ------> else  if (pid_DATA && rx_valid && rx_active && !rx_err) == 1  ----> DATA

ACTIVE ------> else  if (!rx_active || rx_err || (rx_valid && !(pid_TOKEN || pid_DATA))) == 1  ----> IDLE

TOKEN ------> if  (rx_valid && rx_active && !rx_err) == 1  ----> IDLE

TOKEN ------> else  if (!rx_active || rx_err) == 1  ----> IDLE

DATA ------> if  (rx_valid && rx_active && !rx_err) == 1  ----> IDLE
```

- **In this verilog design we successfully verified our model for if-else with else-if condition.**

❖ FSM for 5th verilog input file - input5.v

Case block of the input5 file which is necessary for designing the FSM :

```
case (state)//synopsys full_case parallel_case
IDLE:
begin
pid_le_sm = 1'b1;
if (rx_valid && rx_active)
next_state = ACTIVE ;
end
ACTIVE:
begin
//Received a ACK from Host
if (pid_ACK && !rx_err) begin
got_pid_ack = 1'b1;
next_state = IDLE ;
end
else if (pid_TOKEN && rx_valid && rx_active && !rx_err) begin
token_le_1 = 1'b1;
next_state = TOKEN ;
end
else if (pid_DATA && rx_valid && rx_active && !rx_err) begin
data_valid_d = 1'b1;
```

```
next_state = DATA ;
end
else if (!rx_active || rx_err || (rx_valid && !(pid_TOKEN || pid_DATA))) begin
seq_err = !rx_err;
if (!rx_active)
next_state = IDLE ;
end
end
TOKEN:
begin
if (rx_valid && rx_active && !rx_err) begin
token_le_2 = 1'b1;
next_state = IDLE ;
end
else if (!rx_active || rx_err) begin
next_state = IDLE ;
end
end
DATA:
begin
if (rx_valid && rx_active && !rx_err)
data_valid_d = 1'b1;
next_state = IDLE ;
end
end
endcase
```

FSM : By code

```
Enter The Name of The Verilog File : input5.v

/*****************************************************************************************************************************

                            Finite State Machine For input5.v

*****************************************************************************************************************************/

IDLE ------> if  (rx_valid && rx_active) == 1  ----> ACTIVE

ACTIVE ------> if  (pid_ACK && !rx_err) == 1  ----> IDLE

ACTIVE ------> else  if (pid_TOKEN && rx_valid && rx_active && !rx_err) == 1  ----> TOKEN

ACTIVE ------> else  if (pid_DATA && rx_valid && rx_active && !rx_err) == 1  ----> DATA

ACTIVE ------> else  if (!rx_active || rx_err || (rx_valid && !(pid_TOKEN || pid_DATA))) == 1  ------> if  (!rx_active) == 1  ----> IDLE

TOKEN ------> if  (rx_valid && rx_active && !rx_err) == 1  ----> IDLE

TOKEN ------> else  if (!rx_active || rx_err) == 1  ----> IDLE

DATA ------> if  (rx_valid && rx_active && !rx_err) == 1  ----> IDLE
```

- **In this design we have successfully verified our code for the Nested if-else loop.**

❖ Limitations :

- The input verilog code should not have any unnecessary indentation ,otherwise data in string would not be stored properly.The front is shown above for different cases.
- The code should have proper spacing between each word ,to code to run.
- We have taken the states from the parameter. So it must follow the following pattern with proper spacing.

  [parameter State0=value, State1=value, ……..... ;

****

Thank You !