**Assignment 4 : FSM**

Roshni Uppala

1011735230

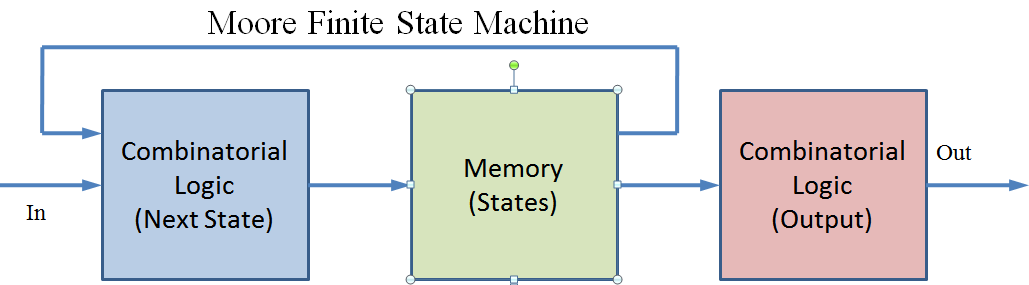
***PROBLEM 1:***

*Objective : To detect a 1110 pattern and output a one (1) with the help of a State transition graph and a verilog module for a Moore type FSM .*

Solution ,

**Design Specification :**

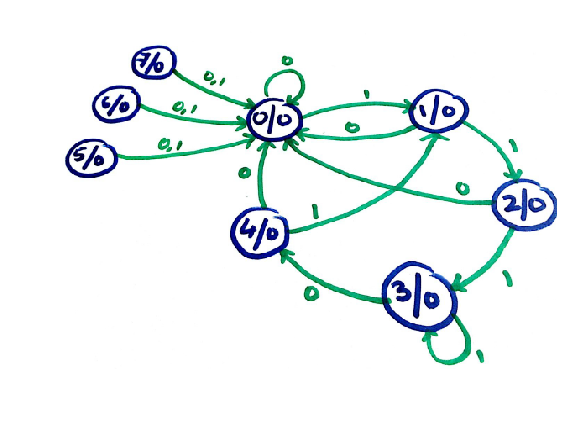
* Inputs : - clk : Clock ; -rst: Rest ; - x : the binary number of the pattern
* Outputs: - state : The state it is in during detection of the pattern ; -y : Output
* Functional Behavior: To detect the pattern 1110 and give an output of one (1).



*Fig 4.1 : Moore FSM*

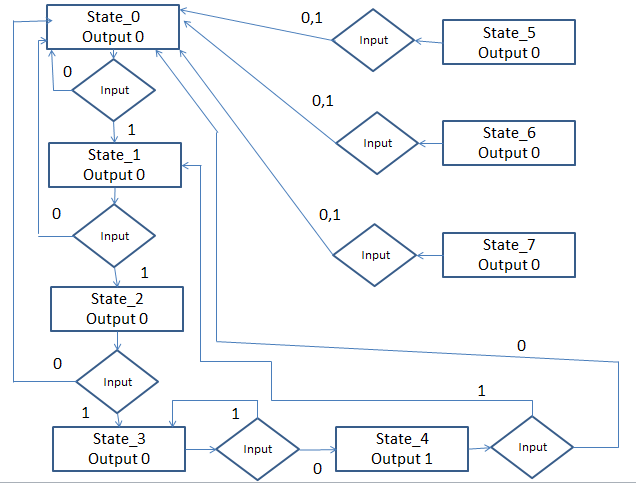
**Design Structure :**

* Inputs :
* Clk : clock
* Rst : reset set to zero
* X : binary digit
* Outputs:
* State : state of the system during detection. It’s a three bit . The states are mentioned as S0 to S7.
* Y: Output of the system ( 0 or 1)
* Functional behavior:
* To detect the pattern of 1110 .
* The different cases are used .
* Cases are set according to the Register transfer language ( RTL )
* State transition graph:



*Fig 4.2 : 1110 Moore STG*

* ASM :



*Fig 4.3 : ASM for 1110 moore fsm*

* Truth Table :

Present state

Next state

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | INPUT | A | B | C | OUTPUT |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

*Table 4.1 : 1110 moore fsm truth table*

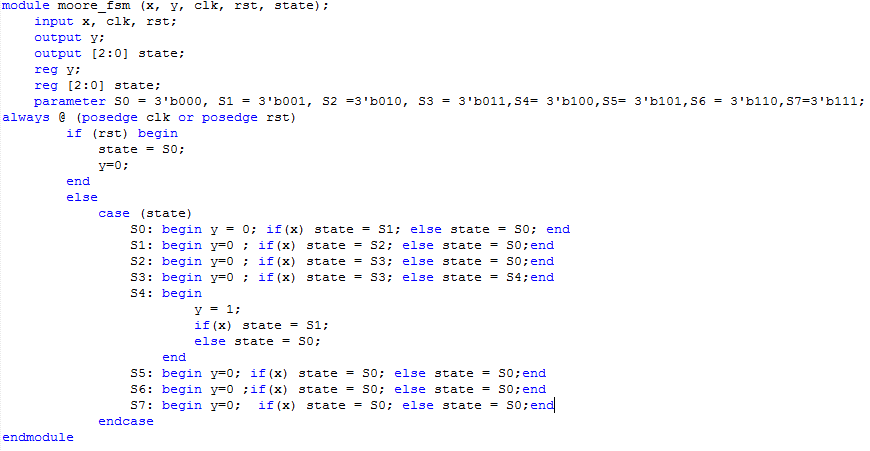
* Register Transfer Language :

|  |  |  |
| --- | --- | --- |
| State | Output | Next state |
| 0 | Y 🡨 0 | (x,!x) /(1/0) |
| 1 | Y 🡨 0 | (x,!x) /(2,0) |
| 2 | Y 🡨 0 | (x,!x) /(3,0) |
| 3 | Y 🡨 0 | (x,!x) /(3,4) |
| 4 | Y 🡨 1 | (x,!x) /(1,0) |
| 5 | Y 🡨 0 | (x,!x) /(0,0) |
| 6 | Y 🡨 0 | (x,!x) /(0,0) |
| 7 | Y 🡨 0 | (x,!x) /(0,0) |

*Table 4.2 : 1110 moore fsm rtl*

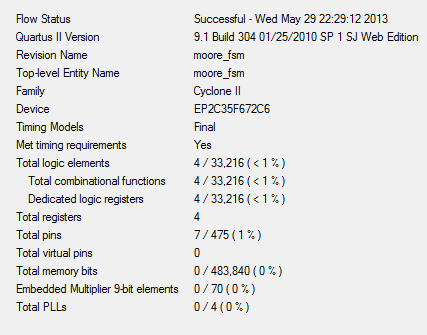
**Design entry :**

* Code :



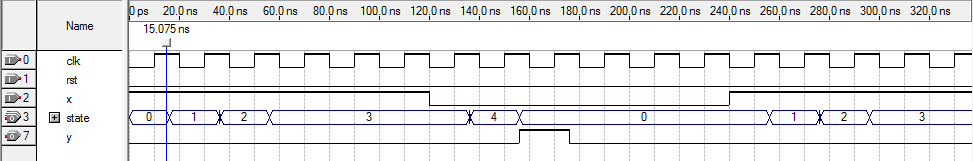
*Fig 4.4 : 1110 moore fsm Verilog code*

* Compilation report:



*Fig 4.5 :1110 moore fsm compilation report*

* Simulation report:



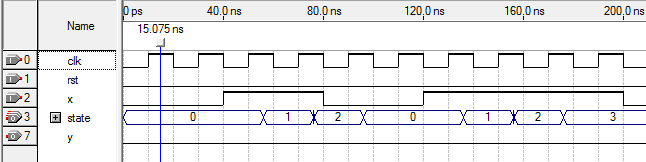
*Fig 4.6 : 1110 moore fsm simulation report*

1110 detected, gives output as 1

**Design verification :**

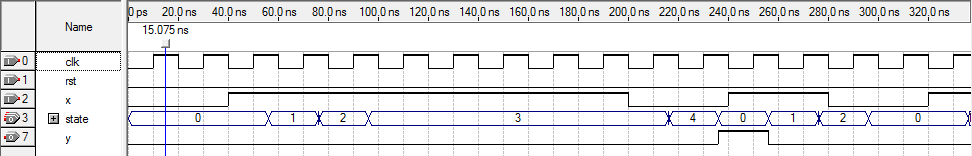
* Test plan :
* Simulation result:

0101



1110 not detected gives output 0

*Fig 4.7 : Test 0101 on moore FSM*



*Fig 4.8 : Test result of 0111 on moore FSM*

1110 not detected gives output 0

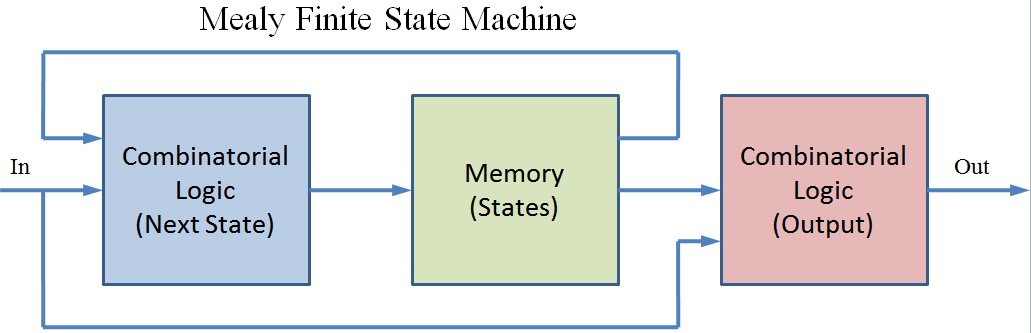
1110 detected gives output 1

***PROBLEM 2 :***

*Objective: To detect a 00011 pattern and output a (1) with the help of a State transition graph and Verilog module for a Mealy type FSM.*

**Design Specification :**

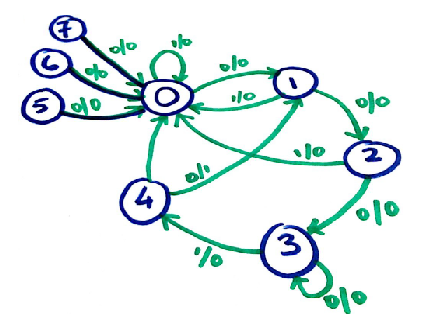
* Inputs : - clk : Clock ; -rst: Rest ; - x : the binary number of the pattern
* Outputs: - state : The state it is in during detection of the pattern ; -y : Output
* Functional Behavior: To detect the pattern 00011 and give an output of one (1).



*Fig 4.9: Mealy fsm*

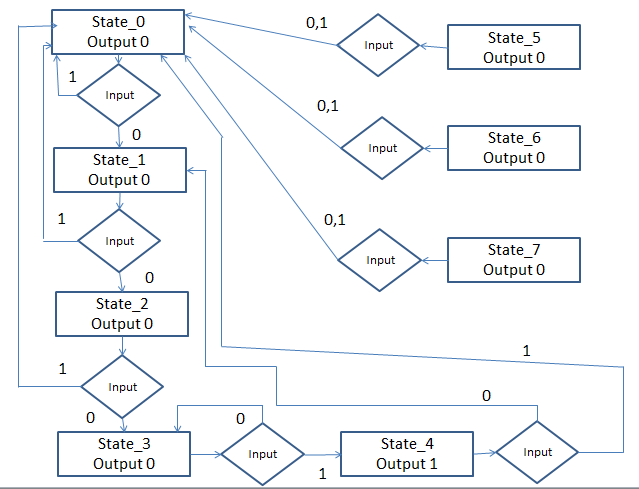
**Design Structure :**

* Inputs :
* Clk : clock
* Rst : reset set to zero
* X : binary digit
* Outputs:
* State : state of the system during detection. It’s a three bit . The states are mentioned as S0 to S7.
* Y: Output of the system ( 0 or 1)
* Functional behavior:
* To detect the pattern of 00011 .
* The different cases are used .
* Cases are set according to the Register transfer language ( RTL )
* STG :



*Fig 4.10 : 00011 Mealy fsm STG*

* ASM :



*Fig 4.11 : 00011 Mealy fsm ASM*

* Truth table:

Next state

Present state

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | INPUT | A | B | C | OUTPUT |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

*Table 4.3: 00011 Mealy fsm truth table*

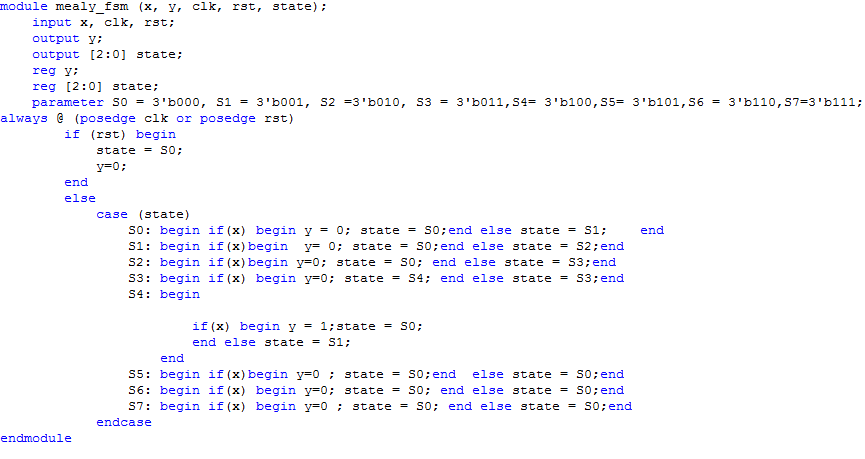
* RTL :

|  |  |  |
| --- | --- | --- |
| State | Output | Next state |
| 0 | Y 🡨 0 | (x,!x) /(1/0) |
| 1 | Y 🡨 0 | (x,!x) /(2,0) |
| 2 | Y 🡨 0 | (x,!x) /(3,0) |
| 3 | Y 🡨 0 | (x,!x) /(3,4) |
| 4 | Y 🡨 1 | (x,!x) /(1,0) |
| 5 | Y 🡨 0 | (x,!x) /(0,0) |
| 6 | Y 🡨 0 | (x,!x) /(0,0) |
| 7 | Y 🡨 0 | (x,!x) /(0,0) |

*Table 4.4 : 00011 Mealy fsm RTL*

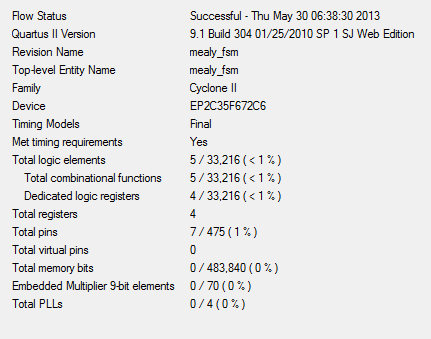
**Design entry :**

* Code :



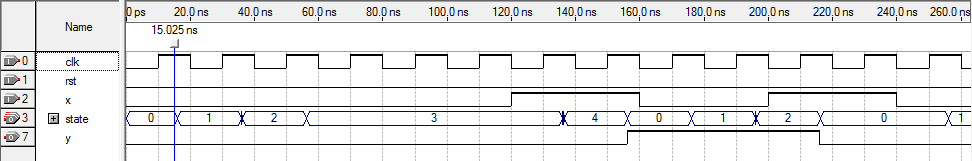
*Fig 4.12 : 00011 mealy fsm Verilog code*

* Compilation report:



*Fig 4.13 : 00011 mealy FSM compilation report*

* Simulation report:

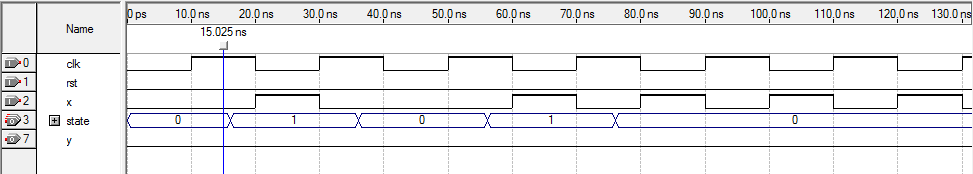


O0011 is detected output is high

*Fig 4.14 : 00011 Mealy simulation report*

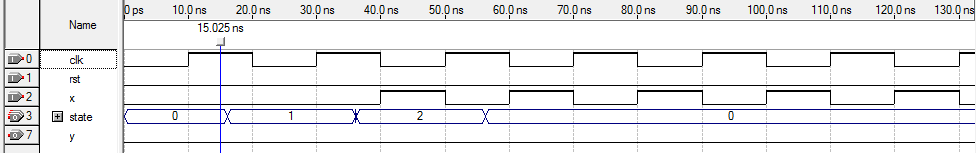
**Design verification :**

* Test plan :
* The design is tested for different patterns by changing the values assigned to x.
* The first figure under simulation result shows the pattern 00100. It satisfies its RTL .
* The second figure resembles the pattern 00001. Its satisfies its RTL .
* Simulation result:



O0011 is not detected output is low

*Fig 4.15 : mealy type test 00100*



O0011 is not detected output is low

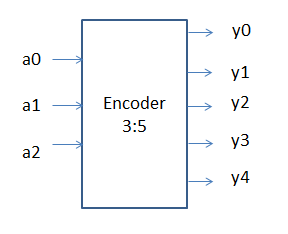
*Fig 4.16: mealy type test 00001*

***Problem 3 :***

*Objective : To design a 3:5 bit encoder.*

**Design Specification :**

* Inputs : - a : 3 bit binary digit
* Outputs : - valid ; - Y: 5 bit binary digit
* Functional Behavior : 3 bit to 5 bit encoder where output = (input\*3)-2



*Fig 4.17 : 3bit to 5 bit*

**Design Structure :**

* Inputs : A 3 bit binary input –a
* Outputs:
* A 5 bit binary output : Y
* A valid
* Functional behavior:
* 3bit input to 5 bit output encoder.
* Output value=(input\*3)-2
* The cases in the program are written basing on the output obtained from the truth table considering output value= (input\*3)-2
* Truth table:

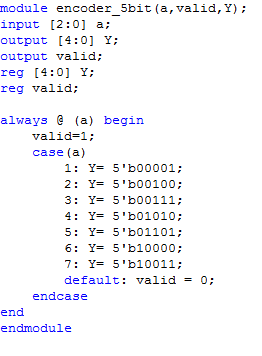
|  |  |  |
| --- | --- | --- |
| **INPUTS** | **OUTPUTS** | **Output value** |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A0** | **A1** | **A2** | **Y0** | **Y1** | **Y2** | **Y3** | **Y4** |  |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 7 |
| 4 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 10 |
| 5 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 13 |
| 6 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 16 |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 19 |

*Table 4.5 : 3bit to 5 bit encoder truth table*

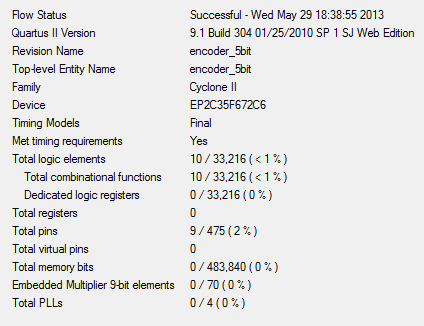
**Design entry :**

* Code :



*Fig 4.18 : 3 bit to 5 bit encoder Verilog code*

* Compilation report:



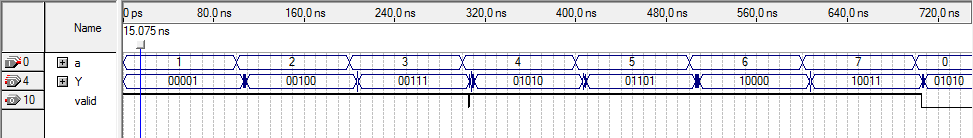
*Fig 4.19 : 3bit to 5 bit compilation report*

**Design verification :**

* Test plan :

The a has input values (3 bit ) as 1,2,3,4,5,6,7. Corresponding output values are cross-checked with the truth table given in table 4.5 and output is 5 bit value.

* Simulation result:



*4.20 : 3bit to 5bit encoder*

For 3 , output = (3\*3)-2= 4

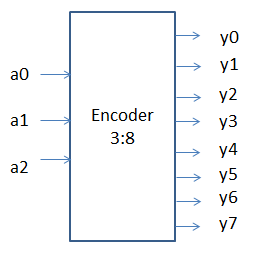
For 7 , output = (7\*3)-2= 19=10011

***Problem 4 :***

*Objective : To design a 3:8 bit encoder.*

**Design Specification :**

* Inputs : - a : 3 bit binary digit
* Outputs : - valid ; - Y: 8 bit binary digit
* Functional Behavior : 3 bit to 8 bit encoder



*Fig 4.21: 3bit to 8 bit encoder*

**Design Structure :**

* Inputs : A 3 bit binary input –a
* Outputs:
* A 5 bit binary output : Y
* A valid
* Functional behavior: 3 bit to 8 bit encoder for the following truth table.
* Truth table:

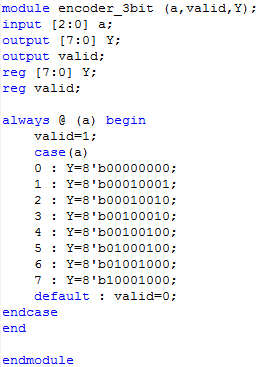
|  |  |
| --- | --- |
| **INPUTS** | **OUTPUTS** |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **A0** | **A1** | **A2** | **Y0** | **Y1** | **Y3** | **Y4** | **Y5** | **Y6** | **Y7** | **Y8** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

*Table 4.6 : 3bit t 8 bit encoder truth table*

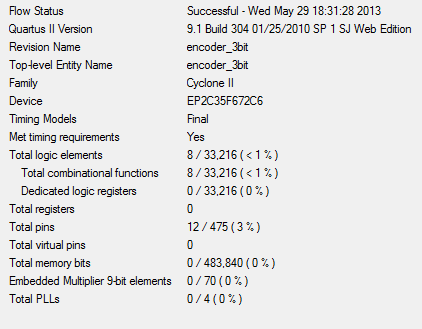
**Design entry :**

* Code :



*Fig 4.22: 3bit to 8 bit encoder Verilog code*

* Compilation report:



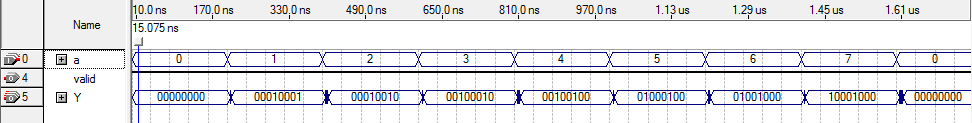
*Fig 4.23: 3bit to 8 bit encoder compilation report*

**Design verification :**

* Test plan :

The input values of a (3 bits) are taken as 0,1,2,3,4,5,6,7. Their respective output values obtained from the simulation are made to match with the table 4.6.

* Simulation result:



*Fig 4.24 : 3bit to 8 bit encoder simulation result*

From truth table 4.6 input of 7 gives output of 10001000

From truth table 4.6 input of 1 gives output of 00010001

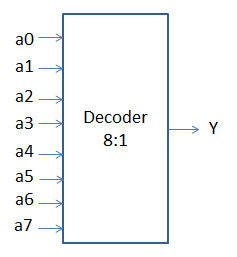
***Problem 5:***

*Objective : To design a 8:1 bit decoder.*

**Design Specification :**

* Inputs : -a: 8 bit input
* Outputs: Y- 1 bit output
* Functional Behavior : 8 bit input to 1 bit binary output decoder by setting output high if two adjacent bits in the input are 1.

Decoder :



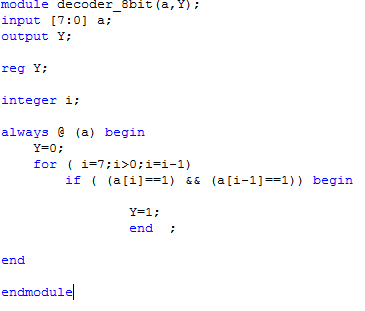
*Fig 4.25 : 8 bit to 1 bit decoder*

**Design Structure :**

* Inputs : A 8 bit binary input –a
* Outputs:
* A 1 bit binary output : Y
* Functional behavior:
* 8 bit to 1 bit decoder.
* Set output high if two adjacent bits in the input are 1

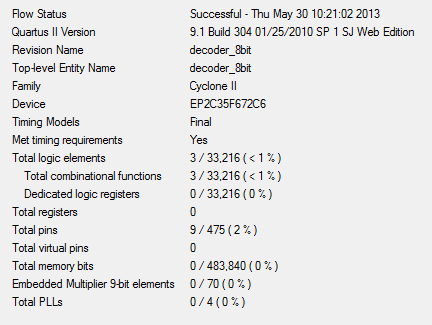
**Design entry :**

* Code :



*Fig 4.26 : 8bit to 1 bit Verilog code*

* Compilation report:



*Fig 4.27: 8bit to 1 bit compilation report*

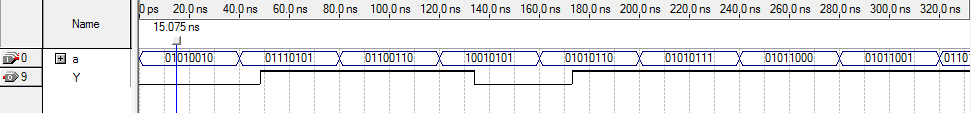
**Design verification :**

* Test plan :
* The input a : 8 bit is set according to the table below. The outputs are very verified with table look up.

|  |  |
| --- | --- |
| INPUT | OUTPUT |
| 01010010 | 0 |
| 01110101 | 1 |
| 01100110 | 1 |
| 10010101 | 0 |

*Table 4.7: Test table for 8 bit to 1 bit decoder*

* Simulation result:



a= 01100110 There are adjacent bits with 1. So output is high.

*Fig 4.28 : 8 bit to 1 bit simulation result*

a= 01010010 There are no adjacent bits with 1. So output is 0