

# FPGA designs with MyHDL



**Meher Krishna Patel**

Created on : October, 2017

Last updated : October, 2018

# Table of contents

<b>Table of contents</b>	<b>i</b>
<b>1 Overview</b>	<b>1</b>
1.1 Introduction	1
1.1.1 Installation and Codes	1
1.1.2 Keywords	1
1.2 First design	2
1.2.1 Define 'and' gate	2
1.2.2 Instantiate 'and' gate	3
1.2.3 Multiple designs in one file	4
1.2.4 Design and conversion codes in the same files	6
1.2.5 Updated port-names for pin assignments using .csv file	7
1.3 Comparator	9
1.3.1 1-bit comparator	9
1.3.2 2-bit comparator	12
1.3.3 Structural modeling	13
1.4 Conclusion	15
<b>2 Visual verification of designs</b>	<b>16</b>
2.1 Introduction	16
2.2 Flip flops	16
2.2.1 Basic D flip flop	16
2.2.2 D-FF with enable port	17
2.3 Mod-m Counter	19
2.4 Clock ticks	21
2.5 Seven segment display	22
2.6 Visual verification of Mod-m counter	25
2.7 Open port in MyHDL	28
2.8 Conclusion	33
<b>3 Testbench</b>	<b>34</b>
3.1 Mod-m counter	34
3.2 Saving data to file	36
3.3 Conversion : MyHDL testbench to HDL testbench	38
3.4 Conclusion	42
<b>4 Finite state machine</b>	<b>43</b>
4.1 Introduction	43
4.2 Rising edge detector	43
4.3 Non overlapping sequence detector : 110	46
4.4 Timed FSM - programmable square wave	50
4.5 Mod-m counter	52
4.6 Conclusion	54

<b>5</b>	<b>More Design examples</b>	<b>55</b>
5.1	Introduction . . . . .	55
5.2	Linear Feedback Shift Register (LFSR) . . . . .	55
5.3	Random Access Memory (RAM) . . . . .	59
5.3.1	Single port RAM . . . . .	59
5.3.2	Dual port RAM . . . . .	61
5.4	Read only memory (ROM) . . . . .	63

# Chapter 1

## Overview

### 1.1 Introduction

In this chapter, we will see various keywords of MyHDL which can be used to create the synthesizable FPGA-designs. We assume the familiarity with the various terms of FPGA designs e.g. signals, process block, always block and reg etc. which are discussed in [Verilog/SystemVerilog/VHDL](#) tutorials. Please refer to [Verilog/SystemVerilog/VHDL](#) tutorials to understand various terms which are used in FPGA designs.

Similar to [Verilog/SystemVerilog/VHDL](#) tutorials, in this chapter, we will implement the ‘2-bit comparator’ using different methods. Please see the Chapter “Overview” of Verilog/VHDL tutorial for the details of the designs.

---

**Important:**

- In this tutorial, the designs of [Verilog/SystemVerilog/VHDL](#)-tutorials are re-implemented using MyHDL.
  - In the comments of MyHDL designs, the corresponding verilog/vhdl-file-names are shown, which can be downloaded from the [website](#).
  - Please see the Verilog/VHDL tutorials for better understanding of the resultant designs.
- 

#### 1.1.1 Installation and Codes

---

**Important:**

- The tutorial is created with MyHDL 1.0 and Python 3.6.2.
  - [Click here](#) to download the Python codes of the tutorial. Run the Python codes to generate the VHDL/Verilog files.
- 

#### 1.1.2 Keywords

Following is the list of MyHDL-keywords which are used in this tutorial. Also relationship between these keywords with Verilog/VHDL are shown in below table,

Keywords	Verilog	VHDL
Signal	reg, wire	signal
bool	reg, wire	std_logic
intbv(0)[N:0]	reg, wire	unsigned
enum	reg, wire	type
if,else	if,else or case	if,else or case
tuple of int	case statement for ROM	case statement for ROM
list of bool	reg	array of std_logic
list of intbv	reg	array of unsigned
def	module	entity
@always	@always	process
@always_comb	@always*	process(all)
@always_seq	@always with initial values of reg/wire	process with initial values of signals
ResetSignal	used with @always_seq to define reset signal	
posedge/negedge	posedge/negedge	rising_edge/falling_edge

## 1.2 First design

In this section, we will implement a simple ‘and gate’ using MyHDL, to see the various components of the MyHDL. This simple example will help us to understand the basic interconnection between the VHDL/Verilog-keywords with MyHDL-keywords.

### 1.2.1 Define ‘and’ gate

- Below is the Python code which implements the ‘and gate’,

```

1  # gateEx.py
2  # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
3
4  from myhdl import *
5
6  # Module : and gate
7  @block
8  def andEx(x, y, z):
9      """ input : x, y
10         output : z
11         """
12
13         # behavior : and gate
14         # i.e. implement and gate using combinational logic
15         @always_comb
16         def and_behave():
17             z.next = x & y # and operation
18
19         # return instances individually
20         # return and_behave
21
22         # this is more convenient as it returns all the instances automatically
23         return instances()

```

**Note:** Above design can be loosely connected with Verilog/VHDL design as follows,

- The first method, i.e. ‘andEx’, is similar to module/entity of the Verilog/VHDL design.
- The second method, i.e. ‘and\_behave’, defines the implementation method i.e. combinational design, sequential design or latches, which is similar to always/architecture block of the Verilog/VHDL designs.

- ‘z.next = x & y’ is same as non-blocking assignments i.e. ‘z <= x & y’ in Verilog or ‘z <= x and y’ in VHDL.

## 1.2.2 Instantiate ‘and’ gate

The above design needs to be instantiated to convert it into VHDL/Verilog code.

- Run following code to convert the Python code to Verilog and VHDL codes,

```
$ python gateEx_convert.py
```

### Note:

- The keyword ‘Signal’ is similar to signal in VHDL or reg/wire in Verilog
- Note that, the ‘x’, ‘y’ and ‘z’ are not signals, but the ports.

```

1  # gateEx_convert.py
2  # convert the gateEx.py to VHDL/Verilog code
3  # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
4
5  from myhdl import *
6  from gateEx import andEx
7
8  x = Signal(bool(0)) # signal of type boolean size 1-bit
9  y = Signal(bool(0))
10 z = Signal(bool(0))
11
12 # convert into Verilog code
13 andEx_verilog = andEx(x, y, z)
14
15 # positional assignments
16 # andEx_verilog = andEx(x=x, y=y, z=z)
17
18 # initial_values = True will initialize the signal
19 # note that there is no signal in this design (all are ports)
20 andEx_verilog.convert(hdl="Verilog", initial_values=True)
21
22 # convert into VHDL code : convert without instantiation
23 andEx(x, y, z).convert(hdl="VHDL", initial_values=True)

```

- The generated Verilog code is shown below,

```

1  // File: andEx.v
2  // Generated by MyHDL 1.0dev
3  // Date: Thu Oct 12 20:34:17 2017
4
5  `timescale 1ns/10ps
6
7  module andEx (
8      x,
9      y,
10     z
11 );
12 // input : x, y
13 // output : z
14
15 input x;
16 input y;
17 output z;
18 wire z;

```

(continues on next page)

(continued from previous page)

```

19
20 assign z = (x & y);
21
22 endmodule

```

- The generated VHDL code is shown below. Note that, a package 'pck\_myhdl\_10.vhd' is also created by MyHDL, which contains various functions in it.

**Note:**

- Do not forget to add the 'pck\_myhdl\_10.vhd' file in the project while synthesizing the code.
- Or comment the line 'use work.pck\_myhdl\_10.all;' from the generated VHDL file, if it is not required.

```

1  -- File: andEx.vhd
2  -- Generated by MyHDL 1.0dev
3  -- Date: Thu Oct 12 20:34:17 2017
4
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.numeric_std.all;
9  use std.textio.all;
10
11 use work.pck_myhdl_10.all;
12
13 entity andEx is
14     port (
15         x: in std_logic;
16         y: in std_logic;
17         z: out std_logic
18     );
19 end entity andEx;
20 -- input : x, y
21 -- output : z
22
23 architecture MyHDL of andEx is
24 begin
25
26 z <= (x and y);
27
28 end architecture MyHDL;

```

### 1.2.3 Multiple designs in one file

- Unlike VHDL/Verilog designs, in MyHDL we can add multiple modules in a single file. In the below code, two modules i.e. 'andEx' and 'xorEx' are added in the single file i.e. 'gateEx.py'.

```

1  # gateEx.py
2  # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
3
4  from myhdl import *
5
6  # Module : and gate
7  @block
8  def andEx(x, y, z):
9      """ input : x, y
10         output : z
11         """

```

(continues on next page)

(continued from previous page)

```

12
13     # behavior : and gate
14     # i.e. implement and gate using combinational logic
15     @always_comb
16     def and_behave():
17         z.next = x & y # and operation
18
19     # return instances individually
20     # return and_behave
21
22     # this is more convenient as it returns all the instances automatically
23     return instances()
24
25 # Module : xor gate
26 @block
27 def xorEx(x, y, z):
28     """ input : x, y
29         output : z
30         """
31
32     # behavior : xor gate
33     # i.e. implement xor gate using combinational logic
34     @always_comb
35     def xor_behave():
36         z.next = x ^ y # xor operation
37
38     # return instances individually
39     # return xor_behave
40
41     # this is more convenient as it returns all the instances automatically
42     return instances()

```

- Then 'gateEx\_convert.py' file is used to generate the Verilog code for these two modules.

```

1 # gateEx_convert.py
2 # convert the gateEx.py to VHDL/Verilog code
3 # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
4
5 from myhdl import *
6 from gateEx import andEx, xorEx
7
8 x = Signal(bool(0)) # signal of type boolean size 1-bit
9 y = Signal(bool(0))
10 z = Signal(bool(0))
11
12 # convert into Verilog code
13 andEx_verilog = andEx(x, y, z)
14
15 # initial_values = True will initialize the signal
16 # note that there is no signal in this design (all are ports)
17 andEx_verilog.convert(hdl="Verilog", initial_values=True)
18
19 # convert into VHDL code : convert without instantiation
20 andEx(x, y, z).convert(hdl="VHDL", initial_values=True)
21
22
23 # convert into Verilog code
24 xorEx_verilog = xorEx(x, y, z)
25
26 # initial_values = True will initialize the signal
27 # note that there is no signal in this design (all are ports)
28 xorEx_verilog.convert(hdl="Verilog", initial_values=True)

```

(continues on next page)



(continued from previous page)

```

29
30 # convert into VHDL code : convert without instantiation
31 xorEx(x, y, z).convert(hdl="VHDL", initial_values=True)

```

### 1.2.4 Design and conversion codes in the same files

It is handy to keep the designs and conversion codes in the same file, rather than writing each piece of code in separate files. In this section, we merge the design and conversion code of the ‘and gate’ and ‘xor gate’ in one file.

- For writing the design and conversion code in the same file, we need to define one boilerplate, i.e. ” if `__name__ == ‘__main__’`”, which tells the python that the entry point of the code is the ‘main()’ function, as shown below.
- If we execute the file ‘gateEx.py’, then the results will be same as previous sections.

```

1  # gateEx.py
2  # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
3
4  from myhdl import *
5
6  # Module : and gate
7  @block
8  def andEx(x, y, z):
9      """ input : x, y
10         output : z
11         """
12
13         # behavior : and gate
14         # i.e. implement and gate using combinational logic
15         @always_comb
16         def and_behave():
17             z.next = x & y # and operation
18
19         # return instances individually
20         # return and_behave
21
22         # this is more convenient as it returns all the instances automatically
23         return instances()
24
25  # Module : xor gate
26  @block
27  def xorEx(x, y, z):
28      """ input : x, y
29         output : z
30         """
31
32         # behavior : xor gate
33         # i.e. implement xor gate using combinational logic
34         @always_comb
35         def xor_behave():
36             z.next = x ^ y # xor operation
37
38         # return instances individually
39         # return xor_behave
40
41         # this is more convenient as it returns all the instances automatically
42         return instances()
43
44
45  def main():

```

(continues on next page)

(continued from previous page)

```

46  x = Signal(bool(0)) # signal of type boolean size 1-bit
47  y = Signal(bool(0))
48  z = Signal(bool(0))
49
50  # convert into Verilog code
51  andEx_verilog = andEx(x, y, z)
52
53  # initial_values = True will initialize the signal
54  # note that there is no signal in this design (all are ports)
55  andEx_verilog.convert(hdl="Verilog", initial_values=True)
56
57  # convert into VHDL code : convert without instantiation
58  andEx(x, y, z).convert(hdl="VHDL", initial_values=True)
59
60
61  # convert into Verilog code
62  xorEx_verilog = xorEx(x, y, z)
63
64  # initial_values = True will initialize the signal
65  # note that there is no signal in this design (all are ports)
66  xorEx_verilog.convert(hdl="Verilog", initial_values=True)
67
68  # convert into VHDL code : convert without instantiation
69  xorEx(x, y, z).convert(hdl="VHDL", initial_values=True)
70
71  if __name__ == '__main__':
72      main()

```

### 1.2.5 Updated port-names for pin assignments using .csv file

In previous section, we implemented the comparator using various methods and converted those designs to Verilog codes. Also, we learn to implement the structural modeling using MyHDL.

#### Note:

- In this section, we will create the top level design with updated port names (according to pin-assignment .csv file, provided in Verilog/VHDL tutorials), so that we need not to change the pin-names in the csv file according to each new design.
- Also, the different signal values (i.e. SW and output value) are assigned to single output port (i.e. LEDG). For this, we need to create one additional 'always' block as shown in the module 'top\_xorEx'.
- Further unlike VHDL/Verilog, for two switches SW[2:0] is used in MyHDL (not SW[1:0]).
- The keyword 'intbv' is used to define the vector-signal instead of 'bool', as the 'bool' keyword is not subscriptable.
- Lastly, if we want to change the port name for 'and gate' as well, then we need to write another top level module for andEx.

- In the below code, a top level design is created for the design in [Section 1.2](#). Following port-mapping is done here,
  - For the 'xor gate', the input ports are connect to SW.
  - The inputs and output of 'xor gate' are connected to LEDG.

```

1  # gateEx.py
2  # the resultant Verilog code is same as Listing 'andEx.v of Verilog tutorial
3
4  from myhdl import *
5
6  # Module : and gate

```

(continues on next page)

(continued from previous page)

```

7 @block
8 def andEx(x, y, z):
9     """ input : x, y
10         output : z
11         """
12
13     # behavior : and gate
14     # i.e. implement and gate using combinational logic
15     @always_comb
16     def and_behave():
17         z.next = x & y # and operation
18
19     # return instances individually
20     # return and_behave
21
22     # this is more convenient as it returns all the instances automatically
23     return instances()
24
25 # Module : xor gate
26 @block
27 def xorEx(x, y, z):
28     """ input : x, y
29         output : z
30         """
31
32     # behavior : xor gate
33     # i.e. implement xor gate using combinational logic
34     @always_comb
35     def xor_behave():
36         z.next = x ^ y # xor operation
37
38     # return instances individually
39     # return xor_behave
40
41     # this is more convenient as it returns all the instances automatically
42     return instances()
43
44
45 # top level entity for 'xor' gate
46 @block
47 def top_xorEx(SW, LEDG):
48     # instantiate xorEx : display only output on green LED (LEDG)
49     xorEx_verilog = xorEx(x=SW(0), y=SW(1), z=LEDG)
50
51     return instances()
52
53 def main():
54     x = Signal(bool(0)) # signal of type boolean size 1-bit
55     y = Signal(bool(0))
56     z = Signal(bool(0))
57
58     switch = Signal(intbv(0)[2:0]) # 2 switches
59     led = Signal(intbv(0)[3:0]) # 3 green LED
60
61     # convert into Verilog code
62     andEx_verilog = andEx(x, y, z)
63
64     # initial_values = True will initialize the signal
65     # note that there is no signal in this design (all are ports)
66     andEx_verilog.convert(hdl="Verilog", initial_values=True)
67

```

(continues on next page)

(continued from previous page)

```

68  # convert into VHDL code : convert without instantiation
69  andEx(x, y, z).convert(hdl="VHDL", initial_values=True)
70
71
72  # convert into Verilog code
73  xorEx_verilog = xorEx(x, y, z)
74
75  # initial_values = True will initialize the signal
76  # note that there is no signal in this design (all are ports)
77  xorEx_verilog.convert(hdl="Verilog", initial_values=True)
78
79  # convert into VHDL code : convert without instantiation
80  xorEx(x, y, z).convert(hdl="VHDL", initial_values=True)
81
82  # modified port name according to pin-assignment file
83  top_xorEx(switch, led).convert(hdl="Verilog", initial_values=True)
84
85  if __name__ == '__main__':
86      main()

```

- The corresponding Verilog code for “xor gate” is shown below. Now, the below can be loaded on the FPGA with the pin-assignment file provided in the Verilog/VHDL tutorials.

```

1  // File: top_xorEx.v
2  // Generated by MyHDL 1.0dev
3  // Date: Thu Oct 12 21:40:22 2017
4
5  `timescale 1ns/10ps
6
7  module top_xorEx (
8      SW,
9      LEDG
10 );
11
12 input [1:0] SW;
13 output [2:0] LEDG;
14 wire [2:0] LEDG;
15
16 assign LEDG = (SW[0] ^ SW[1]);
17
18 endmodule

```

## 1.3 Comparator

In this section, we will implement the 1-bit comparator and the 2-bit comparator with different modeling styles,

### 1.3.1 1-bit comparator

- The ‘@always’ decorator along with sensitivity list can be used for implementing the procedural assignments; whereas ‘@always\_comb’ without sensitivity list can be used to implement the ‘continuous assignment’. In this section, the 1-bit comparator is implemented using procedural and continuous assignments.

---

#### Important:

- The continuous or sequential assignments are inferred automatically based on the logics inside the block. More specifically, the combinational design can be implemented by using ‘sequential statements’ or ‘continuous statements’ based on the logic inside the ‘always\_comb’ block.

- Further, both the designs, i.e. combinational and procedural, are written in one file (comparator\_1\_bit.py), but the corresponding Verilog designs are generated as separate files (comparator\_1\_bit\_proc.v and comparator\_1\_bit\_comb.v).

- Below code will generate the Verilog code with continuous and procedural assignments,

```

1  # comparator_1_bit.py
2
3  # the resultant Verilog code is same as Listing 'comparator1Bit.v
4  # of Verilog tutorial
5
6  from myhdl import *
7
8  # 1-bit comparator using procedural assignment
9  @block
10 def comparator_1_bit_proc(x, y, eq):
11     """ x, y : input
12         eq : output
13
14         eq = 1 when x = y else 0
15     """
16
17     s0 = Signal(bool(0))
18     s1 = Signal(bool(0))
19
20     @always(s0, s1)
21     def comparator_1_bit_behave():
22         s0.next = ~x & ~y
23         s1.next = x & y
24         eq.next = s0 | s1
25
26     return comparator_1_bit_behave
27
28
29 # 1-bit comparator using continuous assignment
30 @block
31 def comparator_1_bit_comb(x, y, eq):
32     """ x, y : input
33         eq : output
34
35         eq = 1 when x = y else 0
36     """
37
38     @always_comb
39     def comparator_1_bit_behave():
40         eq.next = (~x & ~y) | (x & y)
41
42     return comparator_1_bit_behave
43
44 def main():
45     x = Signal(bool(0)) # signal of type boolean size 1 bit
46     y = Signal(bool(0))
47     eq = Signal(bool(0))
48
49     # convert into Verilog code
50     comparator_verilog_procedure = comparator_1_bit_proc(x=x, y=y,
51                                                         eq=eq).convert(hdl="Verilog", initial_values=True)
52     comparator_verilog_continuous = comparator_1_bit_comb(x=x, y=y,
53                                                         eq=eq).convert(hdl="Verilog", initial_values=True)
54
55 if __name__ == '__main__':
56     main()

```

- Two separate verilog files will be generated by above code, which are shown below.

**Note:** Since we set the “initial\_values=True” therefore the signals s0 and s1 are initialized with 0 in the file “comparator\_1\_bit\_proc.v”.

```

1 // File: comparator_1_bit_proc.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 07:52:12 2017
4
5
6 `timescale 1ns/10ps
7
8 module comparator_1_bit_proc (
9     x,
10    y,
11    eq
12 );
13 // x, y : input
14 // eq : output
15 //
16 // eq = 1 when x = y else 0
17
18 input x;
19 input y;
20 output eq;
21 reg eq;
22
23 reg s1 = 0;
24 reg s0 = 0;
25
26 always @(s0, s1) begin: COMPARATOR_1_BIT_PROC_COMPARATOR_1_BIT_BEHAVE
27     s0 <= ((~x) & (~y));
28     s1 <= (x & y);
29     eq <= (s0 | s1);
30 end
31
32 endmodule

```

```

1 // File: comparator_1_bit_comb.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 07:52:12 2017
4
5
6 `timescale 1ns/10ps
7
8 module comparator_1_bit_comb (
9     x,
10    y,
11    eq
12 );
13 // x, y : input
14 // eq : output
15 //
16 // eq = 1 when x = y else 0
17
18 input x;
19 input y;
20 output eq;
21 wire eq;
22
23 assign eq = (((~x) & (~y)) | (x & y));

```

(continues on next page)

(continued from previous page)

```

24
25 endmodule

```

### 1.3.2 2-bit comparator

- Following code implements the two bit comparator,

```

1  # comparator_2_bit.py
2  # the resultant Verilog code is same as Listing 'comparator2Bit.v
3  # of Verilog tutorial
4
5  from myhdl import *
6
7  @block
8  def comparator_2_bit(x, y, eq):
9      """ x, y : input
10         eq : output
11
12         eq = 1 when x = y else 0
13         """
14
15         # 'intbv' is used instead of 'bool' as 'bool' is not subscriptable
16         s = Signal(intbv(0)[4:0])
17
18         @always(s)
19         def comparator_2_bit_behave():
20             s[0].next = ~x[1] & ~x[0] & ~y[1] & ~y[0]
21             s[1].next = ~x[1] & x[0] & ~y[1] & y[0]
22             s[2].next = x[1] & ~x[0] & y[1] & ~y[0]
23             s[3].next = x[1] & x[0] & y[1] & y[0]
24             eq.next = s[0] | s[1] | s[2] | s[3]
25
26         return comparator_2_bit_behave
27
28 def main():
29     # 'intbv' is used instead of 'bool' as 'bool' is not subscriptable
30     x = Signal(intbv(0)[2:0]) # signal of type boolean size 1-bit
31     y = Signal(intbv(0)[2:0])
32     eq = Signal(bool(0))
33
34     # convert into Verilog code
35     comparator_verilog = comparator_2_bit(x=x, y=y, eq=eq)
36     comparator_verilog.convert(hdl="Verilog", initial_values=True)
37
38 if __name__ == '__main__':
39     main()

```

- The resultant Verilog code is shown below,

```

1  // File: comparator_2_bit.v
2  // Generated by MyHDL 1.0dev
3  // Date: Fri Oct 13 08:21:35 2017
4
5
6  `timescale 1ns/10ps
7
8  module comparator_2_bit (
9      x,
10     y,
11     eq

```

(continues on next page)

(continued from previous page)

```

12 );
13 // x, y : input
14 // eq : output
15 //
16 // eq = 1 when x = y else 0
17
18 input [1:0] x;
19 input [1:0] y;
20 output eq;
21 reg eq;
22
23 reg [3:0] s = 0;
24
25 always @(s) begin: COMPARATOR_2_BIT_COMPARATOR_2_BIT_BEHAVE
26     s[0] <= (((~x[1]) & (~x[0])) & (~y[1]) & (~y[0]));
27     s[1] <= (((~x[1]) & x[0]) & (~y[1]) & y[0]);
28     s[2] <= ((x[1] & (~x[0])) & y[1] & (~y[0]));
29     s[3] <= ((x[1] & x[0]) & y[1] & y[0]);
30     eq <= ((s[0] | s[1] | s[2] | s[3]));
31 end
32
33 endmodule

```

### 1.3.3 Structural modeling

In structural modeling the predefined designs are connected together to create the larger system. In this section, we will use two *1-bit comparator* to create a 2-bit comparator.

---

**Note:** Unlike VHDL/Verilog, the structural model defined in the MyHDL does not instantiate the smaller unit but re-implement it, as shown in this section.

---

- Two bit comparator design is shown below,

**Warning:**

- We can not use the ‘intbv’ to define the **output-signal** to use it in the instantiation by “**slicing it**”, i.e. below line can not be synthesized.
- “Signal has multiple driver error” is shown for this, as a part of the slice will be updated by one instantiation, whereas the other part will be updated at other instantiation. In the other words, the output signal will be updated at different ‘always-block’ of Verilog.
- Also, note that for input signal small brackets are used i.e a(0), b(0).

```

s = Signal(intbv(0)[2:0])
eq0 = comparator_1_bit(x=a(0), y=b(0), eq=s(0)) # this will not work

```

```

1 # comparator_2_bit_struct.py
2 # this code is similar to comparator2BitStruct.v of verilog tutorial
3
4 from myhdl import *
5 from comparator_1_bit import comparator_1_bit_proc
6
7 @block
8 def comparator_2_bit_struct(a, b, eq):
9
10     # s = Signal(intbv(0)[2:0]) # this does not work
11     s0 = Signal(bool(0))

```

(continues on next page)



(continued from previous page)

```

12     s1 = Signal(bool(0))
13
14     # instantiation : 1-bit comparator
15     # note square brackets are not used i.e. a(0) and b(0)
16     eq0 = comparator_1_bit_proc(x=a(0), y=b(0), eq=s0)
17     eq1 = comparator_1_bit_proc(x=a(1), y=b(1), eq=s1)
18
19     @always(s0, s1)
20     def comparator_2_bit_behave():
21         if ((s0 == 1) & (s1 == 1)) :
22             eq.next = 1
23         else :
24             eq.next = 0
25     return comparator_2_bit_behave, eq0, eq1
26
27 def main():
28     x = Signal(intbv(0)[2:0])
29     y = Signal(intbv(0)[2:0])
30     eq = Signal(bool(0))
31
32     # convert into Verilog code
33     comparator_verilog = comparator_2_bit_struct(a=x, b=y, eq=eq)
34     comparator_verilog.convert(hdl="Verilog", initial_values=True)
35
36 if __name__ == '__main__' :
37     main()

```

- The resultant Verilog code is shown below.

**Warning:** Note that the 1-bit comparator design is not instantiated here, but reimplemented in the 2-bit comparator (see lines 39-43 and 46-50).

```

1 // File: comparator_2_bit_struct.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 11:57:42 2017
4
5
6 `timescale 1ns/10ps
7
8 module comparator_2_bit_struct (
9     a,
10    b,
11    eq
12 );
13
14
15
16 input [1:0] a;
17 input [1:0] b;
18 output eq;
19 reg eq;
20
21 reg s1 = 0;
22 reg s0 = 0;
23 reg comparator_1_bit_proc_1_s1 = 0;
24 reg comparator_1_bit_proc_1_s0 = 0;
25 reg comparator_1_bit_proc_2_s1 = 0;
26 reg comparator_1_bit_proc_2_s0 = 0;
27
28

```

(continues on next page)

(continued from previous page)

```

29 always @(s0, s1) begin: COMPARATOR_2_BIT_STRUCT_COMPARATOR_2_BIT_BEHAVE
30     if (((s0 == 1) & (s1 == 1))) begin
31         eq <= 1;
32     end
33     else begin
34         eq <= 0;
35     end
36 end
37
38
39 always @(comparator_1_bit_proc_1_s0, comparator_1_bit_proc_1_s1) begin: COMPARATOR_2_BIT_
↪STRUCT_COMPARATOR_1_BIT_PROC_1_COMPARATOR_1_BIT_BEHAVE
40     comparator_1_bit_proc_1_s0 <= ((~a[0]) & (~b[0]));
41     comparator_1_bit_proc_1_s1 <= (a[0] & b[0]);
42     s0 <= (comparator_1_bit_proc_1_s0 | comparator_1_bit_proc_1_s1);
43 end
44
45
46 always @(comparator_1_bit_proc_2_s0, comparator_1_bit_proc_2_s1) begin: COMPARATOR_2_BIT_
↪STRUCT_COMPARATOR_1_BIT_PROC_2_COMPARATOR_1_BIT_BEHAVE
47     comparator_1_bit_proc_2_s0 <= ((~a[1]) & (~b[1]));
48     comparator_1_bit_proc_2_s1 <= (a[1] & b[1]);
49     s1 <= (comparator_1_bit_proc_2_s0 | comparator_1_bit_proc_2_s1);
50 end
51
52 endmodule

```

## 1.4 Conclusion

In this chapter, we saw various features of MyHDL to implement the synthesizable designs. The @always and @always\_comb blocks are used for implementing the continuous assignment, procedural assignment and structural designs. Also, we saw basic differences between the pure Verilog design and the Verilog design generated by the MyHDL. Further, we saw the method to create the top level module according to 'pin-assignment-file'. Lastly, we learn the method by which the design and it's conversion functions can be written in the same file.

# Chapter 2

## Visual verification of designs

### 2.1 Introduction

---

**Note:** In this chapter, a Mod-m counter is implemented and the counting is displayed on the LEDs and seven-segment-display device. Since, we want to visualize the output on these devices, therefore the reduced clock rate is required to visualize the counts.

In this chapter, we will re-implement the designs of the Chapter “Visual verification of designs” in the Verilog/VHDL tutorial. Please see the [Verilog/SystemVerilog/VHDL](#) tutorials for the explanation of the designs.

---

### 2.2 Flip flops

In this section, two types of flip flops are implemented.

#### 2.2.1 Basic D flip flop

- In the following code, the D-FF is implemented which transfer the input value to the output port after the delay of one clock cycle.

```
1  # basic_dff.py
2  # results is same as "basicDff.v" in the Verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def basic_dff(clk, reset_n, d, q):
8      """ clk, reset, d : input
9          q : output
10         """
11
12     @always(clk.posedge, reset_n.negedge)
13     def dff_behave():
14         if reset_n == 0 :
15             q.next = 0
16         else :
17             q.next = d
18     return dff_behave
19
20
```

(continues on next page)

(continued from previous page)

```

21 def main():
22     clk = Signal(bool(0))
23     reset_n = Signal(bool(0))
24     d = Signal(bool(0))
25     q = Signal(bool(0))
26
27     top_dff = basic_dff(clk, reset_n, d, q)
28     top_dff.convert(hdl="Verilog", initial_values=True)
29
30     # function main() is the entry point
31     if __name__ == '__main__':
32         main()

```

- The resultant Verilog code is shown below,

```

1 // File: basic_dff.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 13:56:20 2017
4
5
6 `timescale 1ns/10ps
7
8 module basic_dff (
9     clk,
10    reset_n,
11    d,
12    q
13 );
14 // clk, reset, d : input
15 // q : output
16
17 input clk;
18 input reset_n;
19 input d;
20 output q;
21 reg q;
22
23 always @(posedge clk, negedge reset_n) begin: BASIC_DFF_DFF_BEHAVE
24     if ((reset_n == 0)) begin
25         q <= 0;
26     end
27     else begin
28         q <= d;
29     end
30 end
31
32 endmodule

```

## 2.2.2 D-FF with enable port

- In the below code, the 'enable' port is added to D-FF, so that input value is transferred to output port when the enable signal is high.

```

1 # d_ff.py
2 # results is same as "D_FF.v" in the Verilog tutorial
3
4 from myhdl import *
5
6 @block
7 def dff_unit(clk, reset_n, en, d, q):

```

(continues on next page)

(continued from previous page)

```

8      """ clk, reset, en, d : input
9          q : output
10     """
11
12     @always(clk.posedge, reset_n.negedge)
13     def dff_behave():
14         if reset_n == 0 :
15             q.next = 0
16         # else :
17             # q.next = d
18         elif en == 1 :
19             q.next = d
20     return dff_behave
21
22     # below module is created to change the name of the port
23     # i.e. clk is changed to CLOCK_50
24     @block
25     def d_ff(CLOCK_50, reset_n, en, d, q):
26         dff_top = dff_unit(CLOCK_50, reset_n, en, d, q)
27         return dff_top
28
29     def main():
30         clk = Signal(bool(0))
31         reset_n = Signal(bool(0))
32         en = Signal(bool(0))
33         d = Signal(bool(0))
34         q = ResetSignal(0, active=0, async=True)
35
36         top_dff2 = d_ff(clk, reset_n, en, d, q)
37         top_dff2.convert(hdl="Verilog", initial_values=True)
38
39     # function main() is the entry point
40     if __name__ == '__main__':
41         main()

```

- The resultant Verilog code is shown below,

```

1      // File: d_ff.v
2      // Generated by MyHDL 1.0dev
3      // Date: Fri Oct 13 14:05:54 2017
4
5
6      `timescale 1ns/10ps
7
8      module d_ff (
9          CLOCK_50,
10         reset_n,
11         en,
12         d,
13         q
14     );
15
16
17     input CLOCK_50;
18     input reset_n;
19     input en;
20     input d;
21     output q;
22     reg q;
23
24
25     always @(posedge CLOCK_50, negedge reset_n) begin: D_FF_DFF_UNIT_1_DFF_BEHAVE

```

(continues on next page)

(continued from previous page)

```

26     if ((reset_n == 0)) begin
27         q <= 0;
28     end
29     else if ((en == 1)) begin
30         q <= d;
31     end
32 end
33
34 endmodule

```

## 2.3 Mod-m Counter

In this section, a Mod M counter is implemented. Further, a testbench for the Mod M counter is shown in [Section 3.1](#).

- Following is the Python code for Mod M counter,

Listing 2.1: mod\_m\_counter.py

```

1  # mod_m_counter.py
2  # result is similar to 'modMCounter.v' in the verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def mod_m_counter(clk, reset_n, complete_tick, count, M, N):
8      """ M = max count
9          N = minimum bits required to represent M
10         """
11
12         count_reg = Signal(intbv(0)[N:0])
13         count_next = Signal(intbv(0)[N:0])
14
15         @always(clk.posedge, reset_n.negedge)
16         def logic_reg():
17             if reset_n == 0 :
18                 count_reg.next = 0
19             else :
20                 count_reg.next = count_next
21
22         @always_comb
23         def logic_next():
24             if count_reg == M-1 :
25                 count_next.next = 0
26                 complete_tick.next = 1
27             else :
28                 count_next.next = count_reg + 1
29                 complete_tick.next = 0
30
31         @always_comb
32         def out_val():
33             count.next = count_reg
34
35         return out_val, logic_next, logic_reg
36
37
38 def main():
39     N = 3
40     M = 5

```

(continues on next page)

(continued from previous page)

```

41
42     clk = Signal(bool(0))
43     reset_n = Signal(bool(0))
44     complete_tick = Signal(bool(0))
45     count = Signal(intbv(0)[N:0])
46     mod_m_counter_v = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
47     mod_m_counter_v.convert(hdl="Verilog", initial_values=True)
48
49     if __name__ == '__main__':
50         main()

```

- The resultant Verilog code is shown below,

```

1 // File: mod_m_counter.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 14:17:35 2017
4
5
6 `timescale 1ns/10ps
7
8 module mod_m_counter (
9     clk,
10    reset_n,
11    complete_tick,
12    count
13 );
14 // M = max count
15 // N = minimum bits required to represent M
16
17 input clk;
18 input reset_n;
19 output complete_tick;
20 reg complete_tick;
21 output [2:0] count;
22 wire [2:0] count;
23
24 reg [2:0] count_reg = 0;
25 reg [2:0] count_next = 0;
26
27
28 assign count = count_reg;
29
30
31 always @(count_reg) begin: MOD_M_COUNTER_LOGIC_NEXT
32     if (($signed({1'b0, count_reg}) == (5 - 1))) begin
33         count_next = 0;
34         complete_tick = 1;
35     end
36     else begin
37         count_next = (count_reg + 1);
38         complete_tick = 0;
39     end
40 end
41
42
43 always @(posedge clk, negedge reset_n) begin: MOD_M_COUNTER_LOGIC_REG
44     if ((reset_n == 0)) begin
45         count_reg <= 0;
46     end
47     else begin
48         count_reg <= count_next;
49     end

```

(continues on next page)

(continued from previous page)

```

50 end
51
52 endmodule

```

## 2.4 Clock ticks

To visualize the outputs, e.g. counting etc., we need to reduce the clock rate of the system; otherwise the changes will be too fast to observe.

### Warning:

- The signal 'count' is not used anywhere in the 'clock\_tick.py' but it is defined. Unlike VHDL, we can not keep the port Open in MyHDL. And a warning will be generated for the unused signal.
- But 'open port' can be achieved by using 'None' keyword as shown in [Section 2.7](#); also, it will remove the warning.

- The Python code for generating the clock-ticks with reduced clock-rate are shown below,

```

1  # clock_tick.py
2  # result is similar to 'clockTick.v' in the verilog tutorial
3
4  from myhdl import *
5
6  from mod_m_counter import mod_m_counter
7
8  @block
9  def clock_tick(clk, reset_n, clk_pulse, M, N):
10     count = Signal(intbv(0)[N:0])
11     mod_m_counter_inst = mod_m_counter(clk, reset_n, clk_pulse, count, M, N)
12     return mod_m_counter_inst
13
14  # instantiate clock_tick
15  def main():
16     clk = Signal(bool(0))
17     reset_n = Signal(bool(0))
18     clk_pulse = Signal(bool(0))
19     M = 5000000 # 0.1 sec
20     N = 29
21
22     clock_tick_inst = clock_tick(clk, reset_n, clk_pulse, M, N)
23     clock_tick_inst.convert(hdl="Verilog", initial_values=True)
24
25  if __name__ == '__main__':
26     main()

```

- The corresponding Verilog code is shown below,

```

1  // File: clock_tick.v
2  // Generated by MyHDL 1.0dev
3  // Date: Fri Oct 13 14:23:19 2017
4
5
6  `timescale 1ns/10ps
7
8  module clock_tick (
9     clk,
10    reset_n,
11    clk_pulse

```

(continues on next page)



(continued from previous page)

```

12 );
13
14
15 input clk;
16 input reset_n;
17 output clk_pulse;
18 reg clk_pulse;
19
20 wire [28:0] count;
21 reg [28:0] mod_m_counter_1_count_reg = 0;
22 reg [28:0] mod_m_counter_1_count_next = 0;
23
24
25 assign count = mod_m_counter_1_count_reg;
26
27
28 always @(mod_m_counter_1_count_reg) begin: CLOCK_TICK_MOD_M_COUNTER_1_LOGIC_NEXT
29     if (($signed({1'b0, mod_m_counter_1_count_reg}) == (5000000 - 1))) begin
30         mod_m_counter_1_count_next = 0;
31         clk_pulse = 1;
32     end
33     else begin
34         mod_m_counter_1_count_next = (mod_m_counter_1_count_reg + 1);
35         clk_pulse = 0;
36     end
37 end
38
39
40 always @(posedge clk, negedge reset_n) begin: CLOCK_TICK_MOD_M_COUNTER_1_LOGIC_REG
41     if ((reset_n == 0)) begin
42         mod_m_counter_1_count_reg <= 0;
43     end
44     else begin
45         mod_m_counter_1_count_reg <= mod_m_counter_1_count_next;
46     end
47 end
48
49 endmodule

```

## 2.5 Seven segment display

- Below code convert the 'hexadecimal number' into seven-segment-display format,

```

1  # hex_to_seven_seg.py
2  # result is same as 'hexToSevenSegment.v' and
3  # 'hexToSevenSegment_testCircuit.v' in the Verilog tutorial
4
5  from myhdl import *
6
7  @block
8  def hex_to_seven_seg(hex_num, seven_seg):
9
10     @always(hex_num)
11     def logic():
12         if hex_num == 0:
13             seven_seg.next = 0b1000000
14         elif hex_num == 1 :
15             seven_seg.next = 0b1111001

```

(continues on next page)

(continued from previous page)

```

16     elif hex_num == 2 :
17         seven_seg.next = 0b0100100
18     elif hex_num == 3 :
19         seven_seg.next = 0b0110000
20     elif hex_num == 4 :
21         seven_seg.next = 0b0011001
22     elif hex_num == 5 :
23         seven_seg.next = 0b0010010
24     elif hex_num == 6 :
25         seven_seg.next = 0b0000010
26     elif hex_num == 7 :
27         seven_seg.next = 0b1111000
28     elif hex_num == 8 :
29         seven_seg.next = 0b0000000
30     elif hex_num == 9 :
31         seven_seg.next = 0b0010000
32     elif hex_num == 10 :
33         seven_seg.next = 0b0001000
34     elif hex_num == 11 :
35         seven_seg.next = 0b0000011
36     elif hex_num == 12 :
37         seven_seg.next = 0b1000110
38     elif hex_num == 13 :
39         seven_seg.next = 0b0100001
40     elif hex_num == 14 :
41         seven_seg.next = 0b0000110
42     else :
43         # elif hex_num == 15 :
44         seven_seg.next = 0b0001110
45
46     return logic
47
48     # instantiate hex_to_seven_seg with new port names
49 @block
50 def hex_to_seven_seg_test(SW, HEX0):
51     hex_to_seven_seg_inst = hex_to_seven_seg(SW, HEX0)
52     return hex_to_seven_seg_inst
53
54     # instantiate hex_to_seven_seg_test
55 def main():
56     hex_num = Signal(intbv(0)[4:0])
57     seven_seg = Signal(intbv(0)[7:0])
58
59     hex_ss_v = hex_to_seven_seg_test(hex_num, seven_seg)
60     hex_ss_v.convert(hdl="Verilog", initial_values=True)
61
62
63 if __name__ == '__main__':
64     main()

```

- Below is the resultant Verilog code.

**Note:**

- Note that, the if-else statement of Python code is automatically mapped into case-statement in the Verilog code.
- Load this code to FPGA board and change the SW pattern and see the corresponding numbers on LEDG and seven-segment-display device.

```

1 // File: hex_to_seven_seg_test.v
2 // Generated by MyHDL 1.0dev

```

(continues on next page)

(continued from previous page)

```
3 // Date: Fri Oct 13 14:28:15 2017
4
5
6 `timescale 1ns/10ps
7
8 module hex_to_seven_seg_test (
9     SW,
10    HEXO
11 );
12
13
14 input [3:0] SW;
15 output [6:0] HEXO;
16 reg [6:0] HEXO;
17
18
19
20
21 always @(SW) begin: HEX_TO_SEVEN_SEG_TEST_HEX_TO_SEVEN_SEG_1_LOGIC
22     case (SW)
23         'h0: begin
24             HEXO <= 64;
25         end
26         'h1: begin
27             HEXO <= 121;
28         end
29         'h2: begin
30             HEXO <= 36;
31         end
32         'h3: begin
33             HEXO <= 48;
34         end
35         'h4: begin
36             HEXO <= 25;
37         end
38         'h5: begin
39             HEXO <= 18;
40         end
41         'h6: begin
42             HEXO <= 2;
43         end
44         'h7: begin
45             HEXO <= 120;
46         end
47         'h8: begin
48             HEXO <= 0;
49         end
50         'h9: begin
51             HEXO <= 16;
52         end
53         'ha: begin
54             HEXO <= 8;
55         end
56         'hb: begin
57             HEXO <= 3;
58         end
59         'hc: begin
60             HEXO <= 70;
61         end
62         'hd: begin
63             HEXO <= 33;
```

(continues on next page)

(continued from previous page)

```

64     end
65     'he: begin
66         HEXO <= 6;
67     end
68     default: begin
69         HEXO <= 14;
70     end
71     endcase
72 end
73
74 endmodule

```

## 2.6 Visual verification of Mod-m counter

In this section, we will connect all the previous designs i.e. `mod_m_counter`, `hex_to_seven_seg` and `clock_tick` to count the number upto 'M' and display the result on LEDs and seven-segment-display device.

- Below is the top level entity which connects all the designs together and convert it into Verilog code,

```

1  # mod_m_counter_visual_test.py
2  # result is same as th modMCounter_VisualTest.v of the Verilog tutorial
3
4  from myhdl import *
5
6  from mod_m_counter import mod_m_counter
7  from clock_tick import clock_tick
8  from hex_to_seven_seg import hex_to_seven_seg_test
9
10 @block
11 def mod_m_counter_visual_test(CLOCK_50, reset_n, HEX0, LEDG, M, N):
12     count = Signal(intbv(0)[N:0])
13     clk_pulse = Signal(bool(0))
14     complete_tick = Signal(bool(0))
15
16     # reduced clock rate
17     clock_tick_inst = clock_tick(CLOCK_50, reset_n, clk_pulse, 50000000, 29)
18
19     # reduce clock rate is applied to mod_m_counter
20     mod_m_counter_inst = mod_m_counter(clk_pulse, reset_n,
21         complete_tick, count, M, N)
22
23     # count is send for conversion
24     hex_to_seven_seg_test_inst = hex_to_seven_seg_test(count, HEX0)
25
26     # send count to LEDG
27     @always_comb
28     def led_logic():
29         LEDG.next = count
30
31     return instances()
32
33 def main():
34     CLOCK_50 = Signal(bool(0))
35     reset_n = Signal(bool(0))
36     HEX0 = Signal(intbv(0)[7:0])
37     LEDG = Signal(intbv(0)[4:0])
38     M = 12
39     N = 4
40

```

(continues on next page)

(continued from previous page)

```

41     mod_m_counter_inst = mod_m_counter_visual_test(CLOCK_50, reset_n, HEX0, LEDG, M, N)
42     mod_m_counter_inst.convert(hdl="Verilog", initial_values=True)
43
44     if __name__ == '__main__':
45         main()

```

- Below is the resultant Verilog code. Load this code to FPGA board and see the counting on the seven-segment-display device and LEDs.

```

1 // File: mod_m_counter_visual_test.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:04:14 2017
4
5
6 `timescale 1ns/10ps
7
8 module mod_m_counter_visual_test (
9     CLOCK_50,
10    reset_n,
11    HEX0,
12    LEDG
13 );
14
15
16 input CLOCK_50;
17 input reset_n;
18 output [6:0] HEX0;
19 reg [6:0] HEX0;
20 output [3:0] LEDG;
21 wire [3:0] LEDG;
22
23 reg complete_tick = 0;
24 reg clk_pulse = 0;
25 wire [3:0] count;
26 wire [28:0] clock_tick_1_count;
27 reg [28:0] clock_tick_1_mod_m_counter_1_count_reg = 0;
28 reg [28:0] clock_tick_1_mod_m_counter_1_count_next = 0;
29 reg [3:0] mod_m_counter_2_count_reg = 0;
30 reg [3:0] mod_m_counter_2_count_next = 0;
31
32
33
34
35 assign clock_tick_1_count = clock_tick_1_mod_m_counter_1_count_reg;
36
37
38 always @(clock_tick_1_mod_m_counter_1_count_reg) begin: MOD_M_COUNTER_VISUAL_TEST_CLOCK_
↪TICK_1_MOD_M_COUNTER_1_LOGIC_NEXT
39     if (($signed({1'b0, clock_tick_1_mod_m_counter_1_count_reg}) == (50000000 - 1))) begin
40         clock_tick_1_mod_m_counter_1_count_next = 0;
41         clk_pulse = 1;
42     end
43     else begin
44         clock_tick_1_mod_m_counter_1_count_next = (clock_tick_1_mod_m_counter_1_count_reg ↪
↪+ 1);
45         clk_pulse = 0;
46     end
47 end
48
49
50 always @(posedge CLOCK_50, negedge reset_n) begin: MOD_M_COUNTER_VISUAL_TEST_CLOCK_TICK_1_
↪MOD_M_COUNTER_1_LOGIC_REG

```

(continues on next page)

(continued from previous page)

```

51     if ((reset_n == 0)) begin
52         clock_tick_1_mod_m_counter_1_count_reg <= 0;
53     end
54     else begin
55         clock_tick_1_mod_m_counter_1_count_reg <= clock_tick_1_mod_m_counter_1_count_next;
56     end
57 end
58
59
60
61 assign count = mod_m_counter_2_count_reg;
62
63
64 always @(mod_m_counter_2_count_reg) begin: MOD_M_COUNTER_VISUAL_TEST_MOD_M_COUNTER_2_
↳LOGIC_NEXT
65     if (($signed({1'b0, mod_m_counter_2_count_reg}) == (12 - 1))) begin
66         mod_m_counter_2_count_next = 0;
67         complete_tick = 1;
68     end
69     else begin
70         mod_m_counter_2_count_next = (mod_m_counter_2_count_reg + 1);
71         complete_tick = 0;
72     end
73 end
74
75
76 always @(posedge clk_pulse, negedge reset_n) begin: MOD_M_COUNTER_VISUAL_TEST_MOD_M_
↳COUNTER_2_LOGIC_REG
77     if ((reset_n == 0)) begin
78         mod_m_counter_2_count_reg <= 0;
79     end
80     else begin
81         mod_m_counter_2_count_reg <= mod_m_counter_2_count_next;
82     end
83 end
84
85
86 always @(count) begin: MOD_M_COUNTER_VISUAL_TEST_HEX_TO_SEVEN_SEG_TEST_1_HEX_TO_SEVEN_SEG_
↳1_LOGIC
87     case (count)
88         'h0: begin
89             HEXO <= 64;
90         end
91         'h1: begin
92             HEXO <= 121;
93         end
94         'h2: begin
95             HEXO <= 36;
96         end
97         'h3: begin
98             HEXO <= 48;
99         end
100        'h4: begin
101            HEXO <= 25;
102        end
103        'h5: begin
104            HEXO <= 18;
105        end
106        'h6: begin
107            HEXO <= 2;
108        end

```

(continues on next page)

(continued from previous page)

```

109     'h7: begin
110         HEXO <= 120;
111     end
112     'h8: begin
113         HEXO <= 0;
114     end
115     'h9: begin
116         HEXO <= 16;
117     end
118     'ha: begin
119         HEXO <= 8;
120     end
121     'hb: begin
122         HEXO <= 3;
123     end
124     'hc: begin
125         HEXO <= 70;
126     end
127     'hd: begin
128         HEXO <= 33;
129     end
130     'he: begin
131         HEXO <= 6;
132     end
133     default: begin
134         HEXO <= 14;
135     end
136 endcase
137 end
138
139
140
141 assign LEDG = count;
142
143 endmodule

```

## 2.7 Open port in MyHDL

- We can use the 'None' keyword to avoid connection for the unused port. For this, we need to modify the code for the output signal, which is going to be used as 'optional connection'. In the below code, [Section 2.6](#) is reimplemented.
- First modify the 'mod\_m\_counter.py', in which the output signal 'count' and 'complete\_tick' can be the optional connections; therefore these two signals are written inside the 'if' statement with 'None' keyword,

```

1  # mod_m_counter.py
2
3  from myhdl import *
4
5  @block
6  def mod_m_counter(clk, reset_n, complete_tick, count, M, N):
7      """ M = max count
8          N = minimum bits required to represent M
9      """
10
11     count_reg = Signal(intbv(0)[N:0])
12     count_next = Signal(intbv(0)[N:0])
13
14     @always(clk.posedge, reset_n.negedge)

```

(continues on next page)

(continued from previous page)

```

15 def logic_reg():
16     if reset_n == 0 :
17         count_reg.next = 0
18     else :
19         count_reg.next = count_next
20
21 @always_comb
22 def logic_next():
23     if count_reg == M-1 :
24         count_next.next = 0
25     else :
26         count_next.next = count_reg + 1
27
28
29 # optional complete_tick
30 if complete_tick is not None:
31     @always_comb
32     def complete_tick_logic():
33         if count_reg == M-1 :
34             complete_tick.next = 1
35         else :
36             complete_tick.next = 0
37
38 # optional count
39 if count is not None:
40     @always_comb
41     def out_val():
42         count.next = count_reg
43
44 return instances() # return all instances
45
46
47 def main():
48     N = 3
49     M = 5
50
51     clk = Signal(bool(0))
52     reset_n = Signal(bool(0))
53     complete_tick = Signal(bool(0))
54     count = Signal(intbv(0)[N:0])
55
56     mod_m_counter_v = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
57     mod_m_counter_v.convert(hdl="Verilog", initial_values=True)
58
59 if __name__ == '__main__':
60     main()

```

- Next, modify the 'clock\_tick.py', which is instantiating the 'mod\_m\_counter.py', but not using the 'count' port. Hence keep it open (using None) as below,

```

1 # clock_tick.py
2
3 from myhdl import *
4
5 from mod_m_counter import mod_m_counter
6
7 @block
8 def clock_tick(clk, reset_n, clk_pulse, M, N):
9     # count = Signal(intbv(0)[N:0])
10    # mod_m_counter_inst = mod_m_counter(clk, reset_n, clk_pulse, count, M, N)
11

```

(continues on next page)



(continued from previous page)

```

12     # None for count-port
13     mod_m_counter_inst = mod_m_counter(clk, reset_n, clk_pulse, None, M, N)
14     return mod_m_counter_inst
15
16     # instantiate clock_tick
17     def main():
18         clk = Signal(bool(0))
19         reset_n = Signal(bool(0))
20         clk_pulse = Signal(bool(0))
21         M = 5000000 # 0.1 sec
22         N = 29
23
24         clock_tick_inst = clock_tick(clk, reset_n, clk_pulse, M, N)
25         clock_tick_inst.convert(hdl="Verilog", initial_values=True)
26
27     if __name__ == '__main__':
28         main()

```

- Lastly, update the 'mod\_m\_counter\_visual\_test.py', which is instantiating the 'mod\_m\_counter.py' but not using the port 'complete\_tick',

```

1     # mod_m_counter_visual_test.py
2
3     from myhdl import *
4
5     from mod_m_counter import mod_m_counter
6     from clock_tick import clock_tick
7     from hex_to_seven_seg import hex_to_seven_seg_test
8
9     @block
10    def mod_m_counter_visual_test(CLOCK_50, reset_n, HEX0, LEDG, M, N):
11        count = Signal(intbv(0)[N:0])
12        clk_pulse = Signal(bool(0))
13        # complete_tick = Signal(bool(0))
14
15        # reduced clock rate
16        clock_tick_inst = clock_tick(CLOCK_50, reset_n, clk_pulse, 5000000, 29)
17
18        # reduce clock rate is applied to mod_m_counter
19        # mod_m_counter_inst = mod_m_counter(clk_pulse, reset_n,
20                                           # complete_tick, count, M, N)
21        mod_m_counter_inst = mod_m_counter(clk_pulse, reset_n, None, count, M, N)
22
23        # count is send for conversion
24        hex_to_seven_seg_test_inst = hex_to_seven_seg_test(count, HEX0)
25
26        # send count to LEDG
27        @always_comb
28        def led_logic():
29            LEDG.next = count
30        return instances()
31
32    def main():
33        CLOCK_50 = Signal(bool(0))
34        reset_n = Signal(bool(0))
35        HEX0 = Signal(intbv(0)[7:0])
36        LEDG = Signal(intbv(0)[4:0])
37        M = 12
38        N = 4
39
40        mod_m_counter_inst = mod_m_counter_visual_test(CLOCK_50, reset_n, HEX0, LEDG, M, N)

```

(continues on next page)

(continued from previous page)

```

41     mod_m_counter_inst.convert(hdl="Verilog", initial_values=True)
42
43     if __name__ == '__main__':
44         main()

```

- Now run the 'mod\_m\_counter\_visual\_test.py' and following code will be generated without warning. Load this design on the FPGA board to see the counting on the seven-segment-display device.

```

// File: mod_m_counter_visual_test.v
// Generated by MyHDL 1.0dev
// Date: Fri Oct 13 15:16:53 2017

`timescale 1ns/10ps

module mod_m_counter_visual_test (
    CLOCK_50,
    reset_n,
    HEX0,
    LEDG
);

input CLOCK_50;
input reset_n;
output [6:0] HEX0;
reg [6:0] HEX0;
output [3:0] LEDG;
wire [3:0] LEDG;

reg clk_pulse = 0;
wire [3:0] count;
reg [28:0] clock_tick_1_mod_m_counter_1_count_reg = 0;
reg [28:0] clock_tick_1_mod_m_counter_1_count_next = 0;
reg [3:0] mod_m_counter_2_count_reg = 0;
reg [3:0] mod_m_counter_2_count_next = 0;

always @(posedge CLOCK_50, negedge reset_n) begin: MOD_M_COUNTER_VISUAL_TEST_CLOCK_TICK_1
    ↪MOD_M_COUNTER_1_LOGIC_REG
    if ((reset_n == 0)) begin
        clock_tick_1_mod_m_counter_1_count_reg <= 0;
    end
    else begin
        clock_tick_1_mod_m_counter_1_count_reg <= clock_tick_1_mod_m_counter_1_count_next;
    end
end

always @(clock_tick_1_mod_m_counter_1_count_reg) begin: MOD_M_COUNTER_VISUAL_TEST_CLOCK_
    ↪TICK_1_MOD_M_COUNTER_1_LOGIC_NEXT
    if (($signed({1'b0, clock_tick_1_mod_m_counter_1_count_reg}) == (50000000 - 1))) begin
        clock_tick_1_mod_m_counter_1_count_next = 0;
    end
    else begin
        clock_tick_1_mod_m_counter_1_count_next = (clock_tick_1_mod_m_counter_1_count_reg
    ↪+ 1);
    end
end
end

```

(continues on next page)

(continued from previous page)

```

always @(clock_tick_1_mod_m_counter_1_count_reg) begin: MOD_M_COUNTER_VISUAL_TEST_CLOCK_
↔TICK_1_MOD_M_COUNTER_1_COMPLETE_TICK_LOGIC
    if (($signed({1'b0, clock_tick_1_mod_m_counter_1_count_reg}) == (50000000 - 1))) begin
        clk_pulse = 1;
    end
    else begin
        clk_pulse = 0;
    end
end

always @(posedge clk_pulse, negedge reset_n) begin: MOD_M_COUNTER_VISUAL_TEST_MOD_M_
↔COUNTER_2_LOGIC_REG
    if ((reset_n == 0)) begin
        mod_m_counter_2_count_reg <= 0;
    end
    else begin
        mod_m_counter_2_count_reg <= mod_m_counter_2_count_next;
    end
end

always @(mod_m_counter_2_count_reg) begin: MOD_M_COUNTER_VISUAL_TEST_MOD_M_COUNTER_2_
↔LOGIC_NEXT
    if (($signed({1'b0, mod_m_counter_2_count_reg}) == (12 - 1))) begin
        mod_m_counter_2_count_next = 0;
    end
    else begin
        mod_m_counter_2_count_next = (mod_m_counter_2_count_reg + 1);
    end
end

assign count = mod_m_counter_2_count_reg;

always @(count) begin: MOD_M_COUNTER_VISUAL_TEST_HEX_TO_SEVEN_SEG_TEST_1_HEX_TO_SEVEN_SEG_
↔1_LOGIC
    case (count)
        'h0: begin
            HEXO <= 64;
        end
        'h1: begin
            HEXO <= 121;
        end
        'h2: begin
            HEXO <= 36;
        end
        'h3: begin
            HEXO <= 48;
        end
        'h4: begin
            HEXO <= 25;
        end
        'h5: begin
            HEXO <= 18;
        end
        'h6: begin
            HEXO <= 2;
    end
end

```

(continues on next page)

(continued from previous page)

```
end
'h7: begin
    HEXO <= 120;
end
'h8: begin
    HEXO <= 0;
end
'h9: begin
    HEXO <= 16;
end
'ha: begin
    HEXO <= 8;
end
'hb: begin
    HEXO <= 3;
end
'hc: begin
    HEXO <= 70;
end
'hd: begin
    HEXO <= 33;
end
'he: begin
    HEXO <= 6;
end
default: begin
    HEXO <= 14;
end
endcase
end

assign LEDG = count;

endmodule
```

## 2.8 Conclusion

In this chapter, we learn to reduce the clock rate to visualize the outputs on the display devices. The reduced clock rate is used to display the counting on the seven-segment-display device and LEDs. Further, the structure modeling is used here, which makes the code more manageable and reusable.

# Chapter 3

## Testbench

In this chapter, we write the testbench for the [Listing 2.1](#). This testbench contains several features of MyHDL which are enough to start writing the testbenches. Also, conversion of MyHDL testbench to HDL testbench is discussed.

### 3.1 Mod-m counter

- In this section, we will write a testbench for Mod m counter. For this, the [Listing 2.1](#) is modified as below,

```
1  # mod_m_counter.py
2
3  from myhdl import *
4
5  period = 20 # clk frequency = 50 MHz
6
7  @block
8  def mod_m_counter(clk, reset_n, complete_tick, count, M, N):
9      """ M = max count
10         N = minimum bits required to represent M
11         """
12
13     count_reg = Signal(intbv(0)[N:0])
14     count_next = Signal(intbv(0)[N:0])
15
16     @always(clk.posedge, reset_n.negedge)
17     def logic_reg():
18         if reset_n == 0 :
19             count_reg.next = 0
20         else :
21             count_reg.next = count_next
22
23     @always_comb
24     def logic_next():
25         if count_reg == M-1 :
26             count_next.next = 0
27         else :
28             count_next.next = count_reg + 1
29
30
31     # optional complete_tick
32     if complete_tick is not None:
33         @always_comb
34         def complete_tick_logic():
```

(continues on next page)

(continued from previous page)

```

35         if count_reg == M-1 :
36             complete_tick.next = 1
37         else :
38             complete_tick.next = 0
39
40         # optional count
41         if count is not None:
42             @always_comb
43             def out_val():
44                 count.next = count_reg
45
46         return instances() # return all instances
47
48
49 # testbench
50 @block
51 def mod_m_counter_tb():
52     N = 3
53     M = 5
54
55     clk = Signal(bool(0))
56     reset_n = Signal(bool(0))
57     complete_tick = Signal(bool(0))
58     count = Signal(intbv(0)[N:0])
59
60     mod_m_counter_inst = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
61
62     # int is required
63     @always(delay(int(period/2)))
64     def clk_signal():
65         clk.next = not clk
66
67     @instance # reset signal
68     def reset_signal():
69         reset_n.next = 0
70         yield delay(period)
71         reset_n.next = 1
72
73     return instances()
74
75
76 def main():
77     N = 3
78     M = 5
79
80     clk = Signal(bool(0))
81     reset_n = Signal(bool(0))
82     complete_tick = Signal(bool(0))
83     count = Signal(intbv(0)[N:0])
84
85     mod_m_counter_v = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
86     mod_m_counter_v.convert(hdl="Verilog", initial_values=True)
87
88     # test bench
89     tb = mod_m_counter_tb()
90     tb.config_sim(trace=True)
91     tb.run_sim(15*period) # run for 15 clock cycle
92
93 if __name__ == '__main__':
94     main()

```

- If we run the above code, then a *vcd* file, i.e. `mod_m_counter_tb.vcd`, will be generated, which can be

open in [gtkwave](#). Following is the results saved in the .vcd file,

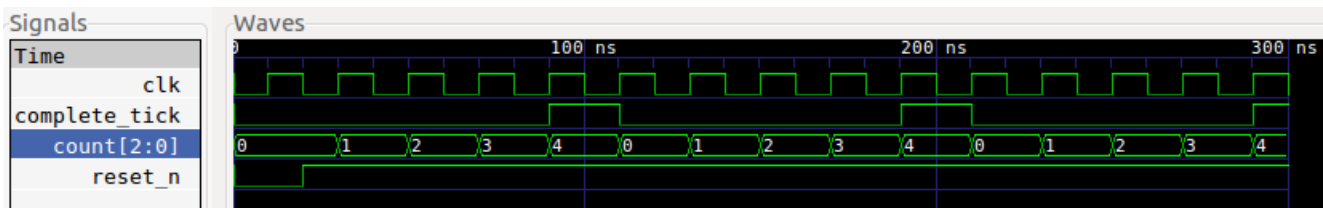


Fig. 3.1: Simulation results of `mod_m_counter.py`

## 3.2 Saving data to file

- Sometimes we want to save the testbench-data in the file for further analysis of the data. In the below listing, the code in the above [Section 3.1](#) is slightly modified to save the data in the file and printed on the screen.

### Important:

- Note that, the signals at Lines 55-61 are declared as `intbv` instead of `bool`, because the `bool` signal is saved as `True/False` which may not be desirable for the analysis of the data.
- Lines 79-93 prints the data on screen and save the data in the file `mod_m_counter.csv`.

```

1  # mod_m_counter.py
2
3  from myhdl import *
4
5  period = 20 # clk frequency = 50 MHz
6
7  @block
8  def mod_m_counter(clk, reset_n, complete_tick, count, M, N):
9      """ M = max count
10         N = minimum bits required to represent M
11         """
12
13         count_reg = Signal(intbv(0)[N:0])
14         count_next = Signal(intbv(0)[N:0])
15
16         @always(clk.posedge, reset_n.negedge)
17         def logic_reg():
18             if reset_n == 0 :
19                 count_reg.next = 0
20             else :
21                 count_reg.next = count_next
22
23         @always_comb
24         def logic_next():
25             if count_reg == M-1 :
26                 count_next.next = 0
27             else :
28                 count_next.next = count_reg + 1
29
30
31         # optional complete_tick
32         if complete_tick is not None:
33             @always_comb
34             def complete_tick_logic():

```

(continues on next page)

(continued from previous page)

```

35         if count_reg == M-1 :
36             complete_tick.next = 1
37         else :
38             complete_tick.next = 0
39
40     # optional count
41     if count is not None:
42         @always_comb
43         def out_val():
44             count.next = count_reg
45
46     return instances() # return all instances
47
48
49 # testbench
50 @block
51 def mod_m_counter_tb():
52     N = 3
53     M = 5
54
55     # 'intbv' is used instead of 'bool' as 'false' will be displayed
56     # in the saved-data for 'bool'.
57     # clk = Signal(bool(0))
58     clk = Signal(intbv(0)[1:0])
59     reset_n = Signal(intbv(0)[1:0])
60     complete_tick = Signal(intbv(0)[1:0])
61     count = Signal(intbv(0)[N:0])
62
63     mod_m_counter_inst = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
64
65
66     # int is required
67     @always(delay(int(period/2)))
68     def clk_signal():
69         clk.next = not clk
70
71
72     @instance # reset signal
73     def reset_signal():
74         reset_n.next = 0
75         yield delay(period)
76         reset_n.next = 1
77
78
79     # print simulation data on screen and file
80     file_data = open("mod_m_counter.csv", 'w') # file for saving data
81     # print header on screen
82     print("{0},{1},{2}".format("reset_n", "complete_tick", "count"))
83     # print header to file
84     print("{0},{1},{2}".format("reset_n", "complete_tick", "count")
85           , file=file_data)
86     # print data on each clock
87     @always(clk.posedge)
88     def print_data():
89         # print on screen
90         print("{0}, {1}, {2}".format(reset_n, complete_tick, count))
91         # print in file
92         print("{0}, {1}, {2}".format(reset_n, complete_tick, count)
93               , file=file_data)
94
95

```

(continues on next page)



(continued from previous page)

```

96     return instances()
97
98
99 def main():
100     N = 3
101     M = 5
102
103     clk = Signal(bool(0))
104     reset_n = Signal(bool(0))
105     complete_tick = Signal(bool(0))
106     count = Signal(intbv(0)[N:0])
107
108     mod_m_counter_v = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
109     mod_m_counter_v.convert(hdl="Verilog", initial_values=True)
110
111     # test bench
112     tb = mod_m_counter_tb()
113     tb.config_sim(trace=True)
114     tb.run_sim(15*period) # run for 15 clock cycle
115
116
117 if __name__ == '__main__':
118     main()

```

- Following is the data which is saved in the file 'mod\_m\_counter.csv'

```

reset_n,complete_tick,count
0, 0, 0
1, 0, 0
1, 0, 1
1, 0, 2
1, 0, 3
1, 1, 4
1, 0, 0
1, 0, 1
1, 0, 2
1, 0, 3
1, 1, 4
1, 0, 0
1, 0, 1
1, 0, 2
1, 0, 3

```

### 3.3 Conversion : MyHDL testbench to HDL testbench

- Some changes are required in above listing to convert the MyHDL testbench into HDL testbench, as shown below. Execute the below code to generate the testbench.

---

**Note:**

- '@instance' decorator is used with "def clk\_signal():", as 'delay' argument is not supported for testbench conversion (Lines 66-70).
  - '.format' option of Python3-print statement is not supported by MyHDL (Lines 91-99).
  - The conversion statement (Line 119) is placed above the simulation statements (Lines 122-123).
- 

```

1 # mod_m_counter.py
2
3 from myhdl import *

```

(continues on next page)

(continued from previous page)

```

4
5 period = 20 # clk frequency = 50 MHz
6
7 @block
8 def mod_m_counter(clk, reset_n, complete_tick, count, M, N):
9     """ M = max count
10         N = minimum bits required to represent M
11         """
12
13     count_reg = Signal(intbv(0)[N:0])
14     count_next = Signal(intbv(0)[N:0])
15
16     @always(clk.posedge, reset_n.negedge)
17     def logic_reg():
18         if reset_n == 0 :
19             count_reg.next = 0
20         else :
21             count_reg.next = count_next
22
23     @always_comb
24     def logic_next():
25         if count_reg == M-1 :
26             count_next.next = 0
27         else :
28             count_next.next = count_reg + 1
29
30
31     # optional complete_tick
32     if complete_tick is not None:
33         @always_comb
34         def complete_tick_logic():
35             if count_reg == M-1 :
36                 complete_tick.next = 1
37             else :
38                 complete_tick.next = 0
39
40     # optional count
41     if count is not None:
42         @always_comb
43         def out_val():
44             count.next = count_reg
45
46     return instances() # return all instances
47
48
49 # testbench
50 @block
51 def mod_m_counter_tb():
52     N = 3
53     M = 5
54
55     # 'intbv' is used instead of 'bool' as 'false' will be displayed
56     # in the saved-data for 'bool'.
57     # clk = Signal(bool(0))
58     clk = Signal(intbv(0)[1:0])
59     reset_n = Signal(intbv(0)[1:0])
60     complete_tick = Signal(intbv(0)[1:0])
61     count = Signal(intbv(0)[N:0])
62
63     mod_m_counter_inst = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
64

```

(continues on next page)

(continued from previous page)

```

65
66 @instance
67 def clk_signal():
68     while True:
69         clk.next = not clk
70         yield delay(period//2)
71
72
73 @instance # reset signal
74 def reset_signal():
75     reset_n.next = 0
76     yield delay(period)
77     reset_n.next = 1
78
79
80 # print simulation data on screen and file
81 file_data = open("mod_m_counter.csv", 'w') # file for saving data
82 # # print header on screen
83 print("{0},{1},{2}".format("reset_n", "complete_tick", "count"))
84 # # print header to file
85 print("{0},{1},{2}".format("reset_n", "complete_tick", "count"),
86       file=file_data)
87 # print data on each clock
88 @always(clk.posedge)
89 def print_data():
90     # print on screen
91     # print.format is not supported in MyHDL 1.0
92     # print("{0}, {1}, {2}".format(reset_n, complete_tick, count))
93     print(reset_n, ",", complete_tick, ",", count, sep='')
94
95     # print in file
96     # print.format is not supported in MyHDL 1.0
97     # print("{0}, {1}, {2}".format(reset_n, complete_tick, count)
98           # , file=file_data)
99     print(reset_n, ",", complete_tick, ",", count, sep='', file=file_data)
100
101
102 return instances()
103
104
105 def main():
106     N = 3
107     M = 5
108
109     clk = Signal(bool(0))
110     reset_n = Signal(bool(0))
111     complete_tick = Signal(bool(0))
112     count = Signal(intbv(0)[N:0])
113
114     mod_m_counter_v = mod_m_counter(clk, reset_n, complete_tick, count, M, N)
115     mod_m_counter_v.convert(hdl="Verilog", initial_values=True)
116
117     # test bench
118     tb = mod_m_counter_tb()
119     tb.convert(hdl="Verilog", initial_values=True)
120     # keep following lines below the 'tb.convert' line
121     # otherwise error will be reported
122     tb.config_sim(trace=True)
123     tb.run_sim(15*period) # run for 15 clock cycle
124
125

```

(continues on next page)

(continued from previous page)

```

126 if __name__ == '__main__':
127     main()

```

- Following is the generated Verilog code for the testbench,

**Note:** In the above listing, the Python code prints the data on screen and save in the file. But in the resultant Verilog code, the data will not be saved in the file (but will be printed on the screen).

```

// File: mod_m_counter_tb.v
// Generated by MyHDL 1.0dev
// Date: Mon Oct 16 11:44:15 2017

`timescale 1ns/10ps

module mod_m_counter_tb (
);

reg [0:0] reset_n = 0;
wire [2:0] count;
reg [0:0] complete_tick = 0;
reg [0:0] clk = 0;
reg [2:0] mod_m_counter_1_count_reg = 0;
reg [2:0] mod_m_counter_1_count_next = 0;

always @(posedge clk, negedge reset_n) begin: MOD_M_COUNTER_TB_MOD_M_COUNTER_1_LOGIC_REG
    if ((reset_n == 0)) begin
        mod_m_counter_1_count_reg <= 0;
    end
    else begin
        mod_m_counter_1_count_reg <= mod_m_counter_1_count_next;
    end
end

always @(mod_m_counter_1_count_reg) begin: MOD_M_COUNTER_TB_MOD_M_COUNTER_1_LOGIC_NEXT
    if (($signed({1'b0, mod_m_counter_1_count_reg}) == (5 - 1))) begin
        mod_m_counter_1_count_next = 0;
    end
    else begin
        mod_m_counter_1_count_next = (mod_m_counter_1_count_reg + 1);
    end
end

always @(mod_m_counter_1_count_reg) begin: MOD_M_COUNTER_TB_MOD_M_COUNTER_1_COMPLETE_TICK_
↳LOGIC
    if (($signed({1'b0, mod_m_counter_1_count_reg}) == (5 - 1))) begin
        complete_tick = 1;
    end
    else begin
        complete_tick = 0;
    end
end

assign count = mod_m_counter_1_count_reg;

initial begin: MOD_M_COUNTER_TB_CLK_SIGNAL
    while (1'b1) begin

```

(continues on next page)

(continued from previous page)

```

        clk <= (!clk);
        # (20 / 2);
    end
end

initial begin: MOD_M_COUNTER_TB_RESET_SIGNAL
    reset_n <= 0;
    # 20;
    reset_n <= 1;
end

always @(posedge clk) begin: MOD_M_COUNTER_TB_PRINT_DATA
    $write("%h", reset_n);
    $write(" ");
    $write(",");
    $write(" ");
    $write("%h", complete_tick);
    $write(" ");
    $write(",");
    $write(" ");
    $write("%h", count);
    $write("\n");
    $write("%h", reset_n);
    $write(" ");
    $write(",");
    $write(" ");
    $write("%h", complete_tick);
    $write(" ");
    $write(",");
    $write(" ");
    $write("%h", count);
    $write("\n");
end

endmodule

```

- Below is the simulation results for the above testbench, which is same as [Fig. 3.1](#) generated by MyHDL testbench.

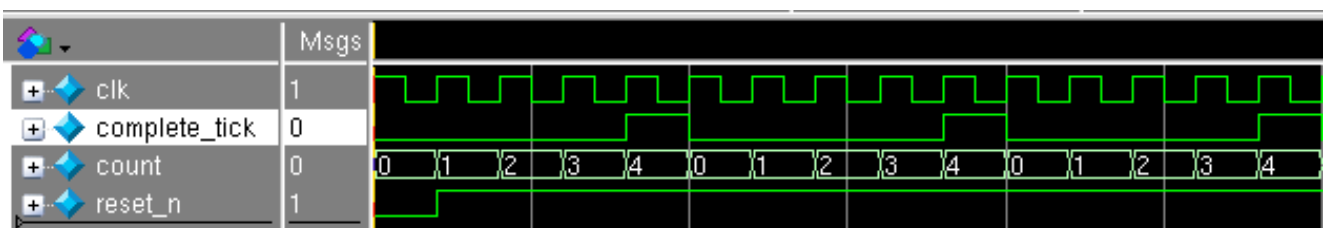


Fig. 3.2: Simulation results of mod\_m\_counter.v

### 3.4 Conclusion

In this chapter, we wrote the testbench for the Mod-m counter. First we wrote a testbench which saves the data in the .vcd file. Next, we learn the method to save the data in the file. Lastly, we modified the MyHDL testbench to convert it into HDL testbench.

# Chapter 4

## Finite state machine

### 4.1 Introduction

---

**Note:**

- In previous chapters, we saw the various examples of combinational and sequential designs. In this chapter, we will learn to use the finite state machines to implement various types of designs.
  - The proper methods for implementing the various kinds of FSM designs are discussed in the Verilog/VHDL tutorials. In this chapter, we will re-implement the designs of the Chapter “Finite state machines” in the Verilog/VHDL tutorial. Please see the [Verilog/SystemVerilog/VHDL](#) tutorials for the explanation of the designs.
- 

### 4.2 Rising edge detector

- In the below code, the rising edge detector is implemented using Mealy and Moore designs,

```
1  # edge_detector.py
2  # the results are same as "edgeDetector.v" in the Verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def edge_detector(clk, reset_n, level, mealy_tick, moore_tick):
8
9      # mealy states
10     mealy_states = enum('zero_mealy', 'one_mealy')
11     state_mealy_reg = Signal(mealy_states.zero_mealy)
12     state_mealy_next = Signal(mealy_states.zero_mealy)
13
14     # moore states
15     moore_states = enum('zero_moore', 'edge_moore', 'one_moore')
16     state_moore_reg = Signal(moore_states.zero_moore)
17     state_moore_next = Signal(moore_states.zero_moore)
18
19     # current state logic
20     @always(clk.posedge, reset_n.negedge)
21     def state_current_logic():
22         if reset_n == 0 :
23             state_moore_reg.next = moore_states.zero_moore
24             state_mealy_reg.next = mealy_states.zero_mealy
```

(continues on next page)

(continued from previous page)

```

25     else :
26         state_moore_reg.next = state_moore_next
27         state_mealy_reg.next = state_mealy_next
28
29     # next state logic : Mealy design
30     @always_comb
31     def state_next_logic_mealy():
32         state_mealy_next.next = state_mealy_reg
33         mealy_tick.next = 0
34         if state_mealy_reg == mealy_states.zero_mealy :
35             if level == 1 :
36                 state_mealy_next.next = mealy_states.one_mealy
37                 mealy_tick.next = 1
38             elif state_mealy_reg == mealy_states.one_mealy:
39                 if level != 1 :
40                     state_mealy_next.next = mealy_states.zero_mealy
41
42     # next state logic : Moore design
43     @always_comb
44     def state_next_logic_moore():
45
46         state_moore_next.next = state_moore_reg
47         moore_tick.next = 0
48         if state_moore_reg == moore_states.zero_moore :
49             if level == 1 :
50                 state_moore_next.next = moore_states.edge_moore
51             elif state_moore_reg == moore_states.edge_moore:
52                 moore_tick.next = 1
53                 if level == 1 :
54                     state_moore_next.next = moore_states.one_moore
55             else :
56                 state_moore_next.next = moore_states.zero_moore
57             elif state_moore_reg == moore_states.one_moore:
58                 if level != 1 :
59                     state_moore_next.next = moore_states.zero_moore
60
61
62     return state_current_logic, state_next_logic_moore, state_next_logic_mealy
63
64 def main():
65     clk = Signal(bool(0))
66     reset_n = Signal(bool(0))
67     level = Signal(bool(0))
68     moore_tick = Signal(bool(0))
69     mealy_tick = Signal(bool(0))
70
71     edge_detector_verilog = edge_detector(clk, reset_n, level, mealy_tick, moore_tick)
72     edge_detector_verilog.convert(hdl="Verilog", initial_values=True)
73
74 if __name__ == '__main__':
75     main()

```

- The resultant Verilog code is shown below,

```

1 // File: edge_detector.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:29:53 2017
4
5
6 `timescale 1ns/10ps
7
8 module edge_detector (

```

(continues on next page)

(continued from previous page)

```
9     clk,
10    reset_n,
11    level,
12    mealy_tick,
13    moore_tick
14 );
15
16
17 input clk;
18 input reset_n;
19 input level;
20 output mealy_tick;
21 reg mealy_tick;
22 output moore_tick;
23 reg moore_tick;
24
25 reg [1:0] state_moore_reg = 2'b00;
26 reg [1:0] state_moore_next = 2'b00;
27 reg [0:0] state_mealy_reg = 1'b0;
28 reg [0:0] state_mealy_next = 1'b0;
29
30
31
32 always @(posedge clk, negedge reset_n) begin: EDGE_DETECTOR_STATE_CURRENT_LOGIC
33     if ((reset_n == 0)) begin
34         state_moore_reg <= 2'b00;
35         state_mealy_reg <= 1'b0;
36     end
37     else begin
38         state_moore_reg <= state_moore_next;
39         state_mealy_reg <= state_mealy_next;
40     end
41 end
42
43
44 always @(state_moore_reg, level) begin: EDGE_DETECTOR_STATE_NEXT_LOGIC_MOORE
45     state_moore_next = state_moore_reg;
46     moore_tick = 0;
47     case (state_moore_reg)
48         2'b00: begin
49             if ((level == 1)) begin
50                 state_moore_next = 2'b01;
51             end
52         end
53         2'b01: begin
54             moore_tick = 1;
55             if ((level == 1)) begin
56                 state_moore_next = 2'b10;
57             end
58             else begin
59                 state_moore_next = 2'b00;
60             end
61         end
62         2'b10: begin
63             if ((level != 1)) begin
64                 state_moore_next = 2'b00;
65             end
66         end
67     endcase
68 end
69
```

(continues on next page)



(continued from previous page)

```

70
71 always @(level, state_mealy_reg) begin: EDGE_DETECTOR_STATE_NEXT_LOGIC_MEALY
72     state_mealy_next = state_mealy_reg;
73     mealy_tick = 0;
74     case (state_mealy_reg)
75         1'b0: begin
76             if ((level == 1)) begin
77                 state_mealy_next = 1'b1;
78                 mealy_tick = 1;
79             end
80         end
81         1'b1: begin
82             if ((level != 1)) begin
83                 state_mealy_next = 1'b0;
84             end
85         end
86     endcase
87 end
88
89 endmodule

```

### 4.3 Non overlapping sequence detector : 110

- In the below code, a sequence detector is implement which detects the sequence '110',

```

1  # sequence_detector.py
2  # result is same as the listing "sequence_detector.v" in verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def sequence_detector(clk, reset_n, x, z_mealy_glitch, z_moore_glitch,
8      z_mealy_glitch_free, z_moore_glitch_free):
9
10
11     # mealy states
12     mealy_states = enum('zero_mealy', 'one_mealy', 'two_mealy', 'three_mealy')
13     state_mealy_reg = Signal(mealy_states.zero_mealy)
14     state_mealy_next = Signal(mealy_states.zero_mealy)
15
16     # moore states
17     moore_states = enum('zero_moore', 'one_moore', 'two_moore', 'three_moore')
18     state_moore_reg = Signal(moore_states.zero_moore)
19     state_moore_next = Signal(moore_states.zero_moore)
20
21     z_moore = Signal(bool(0))
22     z_mealy = Signal(bool(0))
23
24     @always(clk.posedge, reset_n.negedge)
25     def current_state_logic():
26         if reset_n == 0 :
27             state_mealy_reg.next = mealy_states.zero_mealy
28             state_moore_reg.next = moore_states.zero_moore
29         else :
30             state_mealy_reg.next = state_mealy_next
31             state_moore_reg.next = state_moore_next
32
33     # next state mealy logic

```

(continues on next page)

(continued from previous page)

```

34 @always_comb
35 def next_state_logic_mealy():
36     z_mealy.next = 0
37     state_mealy_next.next = state_mealy_reg
38     if state_mealy_reg == mealy_states.zero_mealy:
39         if x == 1 :
40             state_mealy_next.next = mealy_states.one_mealy
41     elif state_mealy_reg == mealy_states.one_mealy :
42         if x == 1 :
43             state_mealy_next.next = mealy_states.two_mealy
44         else :
45             state_mealy_next.next = mealy_states.zero_mealy
46     elif state_mealy_reg == mealy_states.two_mealy :
47         state_mealy_next.next = mealy_states.zero_mealy
48         if x == 0 :
49             z_mealy.next = 1
50         else :
51             state_mealy_next.next = mealy_states.two_mealy
52
53     # next state moore logic
54 @always_comb
55 def next_state_logic_moore():
56     z_moore.next = 0
57     state_moore_next.next = state_moore_reg
58     if state_moore_reg == moore_states.zero_moore:
59         if x == 1 :
60             state_moore_next.next = moore_states.one_moore
61     elif state_moore_reg == moore_states.one_moore :
62         if x == 1 :
63             state_moore_next.next = moore_states.two_moore
64         else :
65             state_moore_next.next = moore_states.zero_moore
66     elif state_moore_reg == moore_states.two_moore :
67         if x == 0 :
68             state_moore_next.next = moore_states.three_moore
69         else :
70             state_moore_next.next = moore_states.two_moore
71     elif state_moore_reg == moore_states.three_moore :
72         z_moore.next = 1
73         if x == 1 :
74             state_moore_next.next = moore_states.zero_moore
75         else :
76             state_moore_next.next = moore_states.one_moore
77
78
79 @always_comb
80 def assign_outputs():
81     z_mealy_glitch.next = z_mealy
82     z_moore_glitch.next = z_moore
83
84 @always(clk.posedge, reset_n.negedge)
85 def glitch_free_output():
86     if reset_n == 1 :
87         z_mealy_glitch_free.next = 0
88         z_moore_glitch_free.next = 0
89     else :
90         z_mealy_glitch_free.next = z_mealy
91         z_moore_glitch_free.next = z_moore
92
93
94 return current_state_logic, next_state_logic_mealy, \

```

(continues on next page)

(continued from previous page)

```

95         next_state_logic_moore, assign_outputs, glitch_free_output
96
97 def main():
98     clk = Signal(bool(0))
99     reset_n = Signal(bool(0))
100    x = Signal(bool(0))
101    z_mealy_glitch = Signal(bool(0))
102    z_moore_glitch = Signal(bool(0))
103    z_mealy_glitch_free = Signal(bool(0))
104    z_moore_glitch_free = Signal(bool(0))
105
106    sequence_detector_verilog = sequence_detector(clk, reset_n, x,
107                                                z_mealy_glitch, z_moore_glitch,
108                                                z_mealy_glitch_free, z_moore_glitch_free)
109
110    sequence_detector_verilog.convert(hdl="Verilog", initial_values=True)
111
112 if __name__ == '__main__':
113     main()

```

- The corresponding Verilog design is shown below,

```

1 // File: sequence_detector.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:33:20 2017
4
5
6 `timescale 1ns/10ps
7
8 module sequence_detector (
9     clk,
10    reset_n,
11    x,
12    z_mealy_glitch,
13    z_moore_glitch,
14    z_mealy_glitch_free,
15    z_moore_glitch_free
16 );
17
18
19 input clk;
20 input reset_n;
21 input x;
22 output z_mealy_glitch;
23 wire z_mealy_glitch;
24 output z_moore_glitch;
25 wire z_moore_glitch;
26 output z_mealy_glitch_free;
27 reg z_mealy_glitch_free;
28 output z_moore_glitch_free;
29 reg z_moore_glitch_free;
30
31 reg z_moore = 0;
32 reg z_mealy = 0;
33 reg [1:0] state_moore_reg = 2'b00;
34 reg [1:0] state_moore_next = 2'b00;
35 reg [1:0] state_mealy_reg = 2'b00;
36 reg [1:0] state_mealy_next = 2'b00;
37
38
39
40 always @(posedge clk, negedge reset_n) begin: SEQUENCE_DETECTOR_CURRENT_STATE_LOGIC

```

(continues on next page)

(continued from previous page)

```
41     if ((reset_n == 0)) begin
42         state_mealy_reg <= 2'b00;
43         state_moore_reg <= 2'b00;
44     end
45     else begin
46         state_mealy_reg <= state_mealy_next;
47         state_moore_reg <= state_moore_next;
48     end
49 end
50
51
52 always @(x, state_mealy_reg) begin: SEQUENCE_DETECTOR_NEXT_STATE_LOGIC_MEALY
53     z_mealy = 0;
54     state_mealy_next = state_mealy_reg;
55     case (state_mealy_reg)
56         2'b00: begin
57             if ((x == 1)) begin
58                 state_mealy_next = 2'b01;
59             end
60         end
61         2'b01: begin
62             if ((x == 1)) begin
63                 state_mealy_next = 2'b10;
64             end
65             else begin
66                 state_mealy_next = 2'b00;
67             end
68         end
69         2'b10: begin
70             state_mealy_next = 2'b00;
71             if ((x == 0)) begin
72                 z_mealy = 1;
73             end
74             else begin
75                 state_mealy_next = 2'b10;
76             end
77         end
78     endcase
79 end
80
81
82 always @(x, state_moore_reg) begin: SEQUENCE_DETECTOR_NEXT_STATE_LOGIC_MOORE
83     z_moore = 0;
84     state_moore_next = state_moore_reg;
85     case (state_moore_reg)
86         2'b00: begin
87             if ((x == 1)) begin
88                 state_moore_next = 2'b01;
89             end
90         end
91         2'b01: begin
92             if ((x == 1)) begin
93                 state_moore_next = 2'b10;
94             end
95             else begin
96                 state_moore_next = 2'b00;
97             end
98         end
99         2'b10: begin
100             if ((x == 0)) begin
101                 state_moore_next = 2'b11;
```

(continues on next page)

(continued from previous page)

```

102         end
103         else begin
104             state_moore_next = 2'b10;
105         end
106     end
107     2'b11: begin
108         z_moore = 1;
109         if ((x == 1)) begin
110             state_moore_next = 2'b00;
111         end
112         else begin
113             state_moore_next = 2'b01;
114         end
115     end
116 endcase
117 end
118
119
120
121 assign z_mealy_glitch = z_mealy;
122 assign z_moore_glitch = z_moore;
123
124
125 always @(posedge clk, negedge reset_n) begin: SEQUENCE_DETECTOR_GLITCH_FREE_OUTPUT
126     if ((reset_n == 1)) begin
127         z_mealy_glitch_free <= 0;
128         z_moore_glitch_free <= 0;
129     end
130     else begin
131         z_mealy_glitch_free <= z_mealy;
132         z_moore_glitch_free <= z_moore;
133     end
134 end
135
136 endmodule

```

## 4.4 Timed FSM - programmable square wave

- In the below code, the square wave is generated with programmable on/off time,

```

1  # square_wave_ex.py
2  # results are same as Listing "square_wave_ex.v" in Verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def square_wave_ex(clk, reset_n, s_wave, on_time, off_time, N):
8      states = enum('on_state', 'off_state')
9      state_reg = Signal(states.on_state)
10     state_next = Signal(states.on_state)
11     t = Signal(intbv(0)[N:0])
12
13     @always(clk.posedge, reset_n.negedge)
14     def current_state_logic():
15         if reset_n == 0 :
16             state_reg.next = states.off_state
17         else :
18             state_reg.next = state_next

```

(continues on next page)

(continued from previous page)

```

19
20     @always(clk.posedge, reset_n.negedge)
21     def timer_logic():
22         if state_reg != state_next :
23             t.next = 0
24         else :
25             t.next = t + 1
26
27     @always_comb
28     def next_state_logic():
29         if state_reg == states.off_state :
30             s_wave.next = 0
31             if t == off_time - 1 :
32                 state_next.next = states.on_state
33             else :
34                 state_next.next = states.off_state
35         if state_reg == states.on_state :
36             s_wave.next = 1
37             if t == on_time - 1 :
38                 state_next.next = states.off_state
39             else :
40                 state_next.next = states.on_state
41
42     return current_state_logic, timer_logic, next_state_logic
43
44 def main():
45     N = 4
46     on_time = 5
47     off_time = 3
48     clk = Signal(bool(0))
49     reset_n = Signal(bool(0))
50     s_wave = Signal(bool(0))
51
52     square_wave_verilog = square_wave_ex(clk, reset_n,
53                                         s_wave, on_time, off_time, N)
54
55     square_wave_verilog.convert(hdl="Verilog", initial_values=True)
56
57 if __name__ == '__main__':
58     main()

```

- Following is the generated Verilog code,

```

1 // File: square_wave_ex.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:39:35 2017
4
5
6 `timescale 1ns/10ps
7
8 module square_wave_ex (
9     clk,
10    reset_n,
11    s_wave
12 );
13
14
15 input clk;
16 input reset_n;
17 output s_wave;
18 reg s_wave;
19

```

(continues on next page)

(continued from previous page)

```

20 reg [3:0] t = 0;
21 reg [0:0] state_reg = 1'b0;
22 reg [0:0] state_next = 1'b0;
23
24
25
26 always @(posedge clk, negedge reset_n) begin: SQUARE_WAVE_EX_CURRENT_STATE_LOGIC
27     if ((reset_n == 0)) begin
28         state_reg <= 1'b1;
29     end
30     else begin
31         state_reg <= state_next;
32     end
33 end
34
35
36 always @(posedge clk, negedge reset_n) begin: SQUARE_WAVE_EX_TIMER_LOGIC
37     if ((state_reg != state_next)) begin
38         t <= 0;
39     end
40     else begin
41         t <= (t + 1);
42     end
43 end
44
45
46 always @(t, state_reg) begin: SQUARE_WAVE_EX_NEXT_STATE_LOGIC
47     if ((state_reg == 1'b1)) begin
48         s_wave = 0;
49         if (($signed({1'b0, t}) == (3 - 1))) begin
50             state_next = 1'b0;
51         end
52     else begin
53         state_next = 1'b1;
54     end
55 end
56     if ((state_reg == 1'b0)) begin
57         s_wave = 1;
58         if (($signed({1'b0, t}) == (5 - 1))) begin
59             state_next = 1'b1;
60         end
61     else begin
62         state_next = 1'b0;
63     end
64 end
65 end
66
67 endmodule

```

## 4.5 Mod-m counter

- In the below code, the Mod-m counter is implemented using FSM method,

```

1 # counter_ex.py
2 # results are same as the Listing 'counter_ex.v' in Verilog tutorial
3
4 from myhdl import *
5

```

(continues on next page)

(continued from previous page)

```

6 @block
7 def counter_ex(clk, reset_n, out_moore, M, N):
8     states = enum('start_moore', 'count_moore')
9     state_moore_reg = Signal(states.start_moore)
10    state_moore_next = Signal(states.start_moore)
11
12    count_moore_reg = Signal(intbv(0)[N:0])
13    count_moore_next = Signal(intbv(0)[N:0])
14
15    @always(clk.posedge, reset_n.negedge)
16    def current_state_logic():
17        if reset_n == 0 :
18            state_moore_reg.next = states.start_moore
19            count_moore_reg.next = 0
20        else :
21            state_moore_reg.next = state_moore_next
22            count_moore_reg.next = count_moore_next
23
24    @always_comb
25    def next_state_logic():
26        if state_moore_reg == states.start_moore :
27            count_moore_next.next = 0
28            state_moore_next.next = states.count_moore
29        elif state_moore_reg == states.count_moore :
30            count_moore_next.next = count_moore_reg + 1
31            if count_moore_reg + 1 == M - 1 :
32                state_moore_next.next = states.start_moore
33            else :
34                state_moore_next.next = states.count_moore
35
36    @always_comb
37    def assign_output_logic():
38        out_moore.next = count_moore_reg
39
40    return current_state_logic, next_state_logic, assign_output_logic
41
42 def main():
43     M = 6
44     N = 4
45     clk = Signal(bool(0))
46     reset_n = Signal(bool(0))
47     out_moore = Signal(bool(0))
48
49     counter_ex_verilog = counter_ex(clk, reset_n, out_moore, M, N)
50     counter_ex_verilog.convert(hdl="Verilog", initial_values=True)
51
52     if __name__ == '__main__':
53         main()

```

- Following is the resultant Verilog code,

```

1 // File: counter_ex.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:43:20 2017
4
5
6 `timescale 1ns/10ps
7
8 module counter_ex (
9     clk,
10    reset_n,
11    out_moore

```

(continues on next page)



(continued from previous page)

```

12 );
13
14
15 input clk;
16 input reset_n;
17 output out_moore;
18 wire out_moore;
19
20 reg [0:0] state_moore_reg = 1'b0;
21 reg [0:0] state_moore_next = 1'b0;
22 reg [3:0] count_moore_reg = 0;
23 reg [3:0] count_moore_next = 0;
24
25
26
27 always @(posedge clk, negedge reset_n) begin: COUNTER_EX_CURRENT_STATE_LOGIC
28     if ((reset_n == 0)) begin
29         state_moore_reg <= 1'b0;
30         count_moore_reg <= 0;
31     end
32     else begin
33         state_moore_reg <= state_moore_next;
34         count_moore_reg <= count_moore_next;
35     end
36 end
37
38
39 always @(count_moore_reg, state_moore_reg) begin: COUNTER_EX_NEXT_STATE_LOGIC
40     case (state_moore_reg)
41         1'b0: begin
42             count_moore_next = 0;
43             state_moore_next = 1'b1;
44         end
45         1'b1: begin
46             count_moore_next = (count_moore_reg + 1);
47             if (((count_moore_reg + 1) == (6 - 1))) begin
48                 state_moore_next = 1'b0;
49             end
50             else begin
51                 state_moore_next = 1'b1;
52             end
53         end
54     endcase
55 end
56
57
58
59 assign out_moore = count_moore_reg;
60
61 endmodule

```

## 4.6 Conclusion

In this chapter, we implemented various FSM designs using MyHDL. We followed the FSM-design-rules which are discussed in [Verilog/SystemVerilog/VHDL](#) tutorials. Further, the resultant Verilog and VHDL codes are exactly same as the Verilog/VHDL tutorial.

# Chapter 5

## More Design examples

### 5.1 Introduction

---

**Note:** In this chapter, some more useful designs are implemented which can be used to build the large systems. Here, we will re-implement the designs of the Chapter “Design examples” in the Verilog/VHDL tutorial. Please see the [Verilog/SystemVerilog/VHDL](#) tutorials for the explanation of the designs.

---

### 5.2 Linear Feedback Shift Register (LFSR)

- In the below code, LFSR is implemented which is used to generate the pseudo random numbers. Further, the generated random number is displayed on seven-segment display device and LEDs.

---

**Note:** In the below code, the generated random number will be displayed on the seven-segment display device and LEDs. Since, the seven segment display device can show the number upto ‘F’ only, therefore the code will work for  $N = 3$  only. While increasing the value of  $N$ , please comment the HEX0 connection from the Python code.

---

```
1  # rand_num_gen.py
2  # result is same as 'rand_num_generator.v' in the Verilog tutorial
3
4  from myhdl import *
5
6  # from mod_m_counter import mod_m_counter
7  from clock_tick import clock_tick
8  from hex_to_seven_seg import hex_to_seven_seg
9
10 @block
11 def rand_num_gen(clk, reset_n, q):
12     N = 3
13     # change the value of feedback_value in feedback_logic according to
14     # the value of N
15     r_reg = Signal(intbv(0)[N+1:0])
16     r_next = Signal(intbv(0)[N+1:0])
17
18     feedback_value = Signal(bool(0))
19
20     @always(clk.posedge, reset_n.negedge)
21     def current_state_logic():
```

(continues on next page)

(continued from previous page)

```

22     if reset_n == 0 :
23         r_reg.next = 1
24     else :
25         r_reg.next = r_next
26
27     @always_comb
28     def feedback_logic():
29         # uncomment correct line based on the value of N
30
31         feedback_value.next = r_reg[3] ^ r_reg[2] ^ r_reg[0] # N = 3
32         # feedback_value.next = r_reg[4] ^ r_reg[3] ^ r_reg[0] # N = 4
33         # feedback_value.next = r_reg[5] ^ r_reg[3] ^ r_reg[0] # N = 5
34         # feedback_value.next = r_reg[9] ^ r_reg[5] ^ r_reg[0] # N = 9
35
36     @always_comb
37     def next_state_logic(): #
38         r_next.next = concat(feedback_value, r_reg[N+1:1])
39
40         ## use above or uncomment below two lines
41         # r_next[N].next = feedback_value
42         # r_next[N:0].next = r_reg[N+1:1]
43
44
45     @always_comb
46     def assign_output():
47         q.next = r_reg
48
49     return current_state_logic, feedback_logic, \
50         assign_output, next_state_logic
51
52 @block
53 def rand_num_gen_top(CLOCK_50, reset_n, LEDG, HEX0):
54     clk_pulse = Signal(bool(0))
55     count = Signal(intbv(0)[4:0])
56
57     # reduced clock rate
58     clock_tick_inst = clock_tick(CLOCK_50, reset_n,
59                                 clk_pulse, 50000000, 29)
60
61     # rand_num_gen with new port names
62     rand_num_gen_verilog = rand_num_gen(clk=clk_pulse,
63                                       reset_n=reset_n, q=count)
64
65     # below will not work for N > 3 as seven-segment can display upto F
66     hex_to_seven_seg_test_inst = hex_to_seven_seg(count, HEX0)
67
68     # send count to LEDG
69     @always_comb
70     def led_logic():
71         LEDG.next = count
72
73     return rand_num_gen_verilog, clock_tick_inst, \
74         hex_to_seven_seg_test_inst, led_logic
75
76
77 def main():
78     N = 3 # set the value of same as rand_num_gen
79     clk = Signal(bool(0))
80     reset_n = Signal(bool(0))
81     q = Signal(intbv(0)[N+1:0])
82

```

(continues on next page)

(continued from previous page)

```

83  # HEX0 will work for N = 3 only, as seven segment can display till F only
84  HEX0 = Signal(intbv(0)[7:0])
85
86  # rand_num_gen for simulation
87  rand_num_gen_verilog = rand_num_gen(clk, reset_n, q)
88  rand_num_gen_verilog.convert(hdl="Verilog", initial_values=True)
89
90  # rand_num_gen for visual testing
91  rand_num_gen_top_verilog = rand_num_gen_top(clk, reset_n, q, HEX0)
92  rand_num_gen_top_verilog.convert(hdl="Verilog", initial_values=True)
93
94  if __name__ == '__main__':
95      main()

```

- Following is the resultant Verilog code generate by the instance “rand\_num\_gen\_top\_verilog” in above listing. Load this design on the FPGA board; and the generated random numbers will be displayed on the seven-segment display device.

```

1  // File: rand_num_gen_top.v
2  // Generated by MyHDL 1.0dev
3  // Date: Fri Oct 13 15:57:19 2017
4
5
6  `timescale 1ns/10ps
7
8  module rand_num_gen_top (
9      CLOCK_50,
10     reset_n,
11     LEDG,
12     HEX0
13 );
14
15
16 input CLOCK_50;
17 input reset_n;
18 output [3:0] LEDG;
19 wire [3:0] LEDG;
20 output [6:0] HEX0;
21 reg [6:0] HEX0;
22
23 reg clk_pulse = 0;
24 wire [3:0] count;
25 reg [3:0] rand_num_gen_1_r_reg = 0;
26 wire [3:0] rand_num_gen_1_r_next;
27 wire rand_num_gen_1_feedback_value;
28 reg [28:0] clock_tick_1_mod_m_counter_1_count_reg = 0;
29 reg [28:0] clock_tick_1_mod_m_counter_1_count_next = 0;
30
31
32
33 always @(posedge clk_pulse, negedge reset_n) begin: RAND_NUM_GEN_TOP_RAND_NUM_GEN_1_
↪CURRENT_STATE_LOGIC
34     if ((reset_n == 0)) begin
35         rand_num_gen_1_r_reg <= 1;
36     end
37     else begin
38         rand_num_gen_1_r_reg <= rand_num_gen_1_r_next;
39     end
40 end
41
42

```

(continues on next page)

(continued from previous page)

```

43
44 assign rand_num_gen_1_feedback_value = ((rand_num_gen_1_r_reg[3] ^ rand_num_gen_1_r_
↳reg[2]) ^ rand_num_gen_1_r_reg[0]);
45
46
47
48 assign count = rand_num_gen_1_r_reg;
49
50
51
52 assign rand_num_gen_1_r_next = {rand_num_gen_1_feedback_value, rand_num_gen_1_r_reg[(3 +
↳1)-1:1]};
53
54
55 always @(posedge CLOCK_50, negedge reset_n) begin: RAND_NUM_GEN_TOP_CLOCK_TICK_1_MOD_M_
↳COUNTER_1_LOGIC_REG
56     if ((reset_n == 0)) begin
57         clock_tick_1_mod_m_counter_1_count_reg <= 0;
58     end
59     else begin
60         clock_tick_1_mod_m_counter_1_count_reg <= clock_tick_1_mod_m_counter_1_count_next;
61     end
62 end
63
64
65 always @(clock_tick_1_mod_m_counter_1_count_reg) begin: RAND_NUM_GEN_TOP_CLOCK_TICK_1_MOD_
↳M_COUNTER_1_LOGIC_NEXT
66     if (($signed({1'b0, clock_tick_1_mod_m_counter_1_count_reg}) == (50000000 - 1))) begin
67         clock_tick_1_mod_m_counter_1_count_next = 0;
68     end
69     else begin
70         clock_tick_1_mod_m_counter_1_count_next = (clock_tick_1_mod_m_counter_1_count_reg
↳+ 1);
71     end
72 end
73
74
75 always @(clock_tick_1_mod_m_counter_1_count_reg) begin: RAND_NUM_GEN_TOP_CLOCK_TICK_1_MOD_
↳M_COUNTER_1_COMPLETE_TICK_LOGIC
76     if (($signed({1'b0, clock_tick_1_mod_m_counter_1_count_reg}) == (50000000 - 1))) begin
77         clk_pulse = 1;
78     end
79     else begin
80         clk_pulse = 0;
81     end
82 end
83
84
85 always @(count) begin: RAND_NUM_GEN_TOP_HEX_TO_SEVEN_SEG_1_LOGIC
86     case (count)
87         'h0: begin
88             HEXO <= 64;
89         end
90         'h1: begin
91             HEXO <= 121;
92         end
93         'h2: begin
94             HEXO <= 36;
95         end
96         'h3: begin
97             HEXO <= 48;

```

(continues on next page)

(continued from previous page)

```

98     end
99     'h4: begin
100         HEXO <= 25;
101     end
102     'h5: begin
103         HEXO <= 18;
104     end
105     'h6: begin
106         HEXO <= 2;
107     end
108     'h7: begin
109         HEXO <= 120;
110     end
111     'h8: begin
112         HEXO <= 0;
113     end
114     'h9: begin
115         HEXO <= 16;
116     end
117     'ha: begin
118         HEXO <= 8;
119     end
120     'hb: begin
121         HEXO <= 3;
122     end
123     'hc: begin
124         HEXO <= 70;
125     end
126     'hd: begin
127         HEXO <= 33;
128     end
129     'he: begin
130         HEXO <= 6;
131     end
132     default: begin
133         HEXO <= 14;
134     end
135 endcase
136 end
137
138
139
140 assign LEDG = count;
141
142 endmodule

```

## 5.3 Random Access Memory (RAM)

In this section, the ‘single port’ and ‘dual port’ RAMs are implemented.

### 5.3.1 Single port RAM

- In the below code, a single port RAM is implemented,

```

1  # single_port_ram.py
2  # result is same as "single_port_RAM.v" of the Verilog tutorial
3

```

(continues on next page)

(continued from previous page)

```

4 from myhdl import *
5
6 @block
7 def single_port_ram(clk, we, addr, din, dout, addr_width=2, data_width=3) :
8     ram_single_port = [Signal(intbv(0)[addr_width:0])
9                        for i in range(data_width)]
10
11     @always(clk.posedge)
12     def write_logic():
13         """ write data to address 'addr' """
14         if we == 1 :
15             ram_single_port[addr].next = din
16
17     @always_comb
18     def read_logic():
19         """ read data from address 'addr' """
20         dout.next = ram_single_port[addr]
21
22     return read_logic, write_logic
23
24 def main():
25     addr_width = 3
26     data_width = 3
27     clk = Signal(bool(0))
28     we = Signal(bool(0))
29     addr = Signal(intbv(0)[addr_width:0])
30     din = Signal(intbv(0)[data_width:0])
31     dout = Signal(intbv(0)[data_width:0])
32
33     single_port_ram_verilog = single_port_ram(clk, we, addr,
34                                               din, dout, addr_width, data_width)
35
36     single_port_ram_verilog.convert(hdl="Verilog", initial_values=True)
37
38 if __name__ == '__main__' :
39     main()

```

- Following is the resultant Verilog code,

```

1 // File: single_port_ram.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 15:59:53 2017
4
5
6 `timescale 1ns/10ps
7
8 module single_port_ram (
9     clk,
10    we,
11    addr,
12    din,
13    dout
14 );
15
16
17 input clk;
18 input we;
19 input [2:0] addr;
20 input [2:0] din;
21 output [2:0] dout;
22 wire [2:0] dout;
23

```

(continues on next page)

(continued from previous page)

```

24 reg [2:0] ram_single_port [0:3-1];
25
26 initial begin: INITIALIZE_RAM_SINGLE_PORT
27     integer i;
28     for(i=0; i<3; i=i+1) begin
29         ram_single_port[i] = 0;
30     end
31 end
32
33
34
35 // read data from address 'addr'
36
37 assign dout = ram_single_port[addr];
38
39 // write data to address 'addr'
40 always @(posedge clk) begin: SINGLE_PORT_RAM_WRITE_LOGIC
41     if ((we == 1)) begin
42         ram_single_port[addr] <= din;
43     end
44 end
45
46 endmodule

```

### 5.3.2 Dual port RAM

- Following code implements a dual port RAM,

```

1  # dual_port_ram.py
2  # result is same as "dual_port_RAM.v" of the Verilog tutorial
3
4  from myhdl import *
5
6  @block
7  def dual_port_ram(clk, we, addr_rd, addr_wr, din, dout,
8                  addr_width=2, data_width=3) :
9
10     ram_dual_port = [Signal(intbv(0)[addr_width:0])
11                     for i in range(data_width)]
12
13     @always(clk.posedge)
14     def write_logic():
15         """ write data to address 'addr' """
16         if we == 1 :
17             ram_dual_port[addr_wr].next = din
18
19     @always_comb
20     def read_logic():
21         """ read data from address 'addr' """
22         dout.next = ram_dual_port[addr_rd]
23
24     return read_logic, write_logic
25
26 def main():
27     addr_width = 3
28     data_width = 3
29     clk = Signal(bool(0))
30     we = Signal(bool(0))
31     addr_rd = Signal(intbv(0)[addr_width:0])

```

(continues on next page)



(continued from previous page)

```

32     addr_wr = Signal(intbv(0)[addr_width:0])
33     din = Signal(intbv(0)[data_width:0])
34     dout = Signal(intbv(0)[data_width:0])
35
36     dual_port_ram_verilog = dual_port_ram(clk, we, addr_rd,
37         addr_wr, din, dout, addr_width, data_width)
38
39     dual_port_ram_verilog.convert(hdl="Verilog", initial_values=True)
40     if __name__ == '__main__':
41         main()

```

- Below is the resultant Verilog code,

```

1 // File: dual_port_ram.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 16:02:12 2017
4
5
6 `timescale 1ns/10ps
7
8 module dual_port_ram (
9     clk,
10    we,
11    addr_rd,
12    addr_wr,
13    din,
14    dout
15 );
16
17
18 input clk;
19 input we;
20 input [2:0] addr_rd;
21 input [2:0] addr_wr;
22 input [2:0] din;
23 output [2:0] dout;
24 wire [2:0] dout;
25
26 reg [2:0] ram_dual_port [0:3-1];
27
28 initial begin: INITIALIZE_RAM_DUAL_PORT
29     integer i;
30     for(i=0; i<3; i=i+1) begin
31         ram_dual_port[i] = 0;
32     end
33 end
34
35
36 // read data from address 'addr'
37
38 assign dout = ram_dual_port[addr_rd];
39
40 // write data to address 'addr'
41 always @(posedge clk) begin: DUAL_PORT_RAM_WRITE_LOGIC
42     if ((we == 1)) begin
43         ram_dual_port[addr_wr] <= din;
44     end
45 end
46
47 endmodule
48

```

## 5.4 Read only memory (ROM)

- In the below code, a ROM is implemented which stores the values for the seven-segment display device. The switches SW provide the address for the ROM and the ROM returns the corresponding pattern for the seven-segment display device.

```

1  # rom_seven_segment.py
2  # the result is same as "ROM_seven_segment.v" in Verilog tutorial
3  from myhdl import *
4
5  @block
6  def rom_seven_segment(addr, dout):
7      # address width = 4
8      # data width = 7
9      data = (
10         0b1000000,
11         0b1111001,
12         0b0100100,
13         0b0110000,
14         0b0011001,
15         0b0010010,
16         0b0000010,
17         0b1111000,
18         0b0000000,
19         0b0010000,
20         0b0001000,
21         0b0000011,
22         0b1000110,
23         0b0100001,
24         0b0000110,
25         0b0001110
26     )
27
28     @always_comb
29     def read_logic():
30         dout.next = data[int(addr)]
31
32     return read_logic
33
34 # top level design with updated port names
35 @block
36 def rom_ss_visual_test(SW, HEX0, LEDG):
37     data = Signal(intbv(0)[7:0])
38     rom_seven_segment_inst = rom_seven_segment(SW, data)
39
40     @always_comb
41     def display_logic():
42         HEX0.next = data
43         LEDG.next = data
44
45     return display_logic, rom_seven_segment_inst
46
47 # verilog conversion
48 def main():
49     SW = Signal(intbv(0)[4:0])
50     LEDG = Signal(intbv(0)[7:0])
51     HEX0 = Signal(intbv(0)[7:0])
52
53     rom_seven_segment_verilog = rom_ss_visual_test(SW, HEX0, LEDG)
54     rom_seven_segment_verilog.convert(hdl="Verilog", initial_values=True)
55

```

(continues on next page)

(continued from previous page)

```

56 if __name__ == '__main__' :
57     main()

```

- The resultant Verilog code is shown below. Load this design to FPGA board and change the switch pattern, and the corresponding HEX value will be shown on seven-segment device.

```

1 // File: rom_ss_visual_test.v
2 // Generated by MyHDL 1.0dev
3 // Date: Fri Oct 13 16:08:29 2017
4
5
6 `timescale 1ns/10ps
7
8 module rom_ss_visual_test (
9     SW,
10    HEX0,
11    LEDG
12 );
13
14
15 input [3:0] SW;
16 output [6:0] HEX0;
17 wire [6:0] HEX0;
18 output [6:0] LEDG;
19 wire [6:0] LEDG;
20
21 reg [6:0] data = 0;
22
23
24
25
26 assign HEX0 = data;
27 assign LEDG = data;
28
29
30 always @(SW) begin: ROM_SS_VISUAL_TEST_ROM_SEVEN_SEGMENT_1_READ_LOGIC
31     case (SW)
32         0: data = 64;
33         1: data = 121;
34         2: data = 36;
35         3: data = 48;
36         4: data = 25;
37         5: data = 18;
38         6: data = 2;
39         7: data = 120;
40         8: data = 0;
41         9: data = 16;
42         10: data = 8;
43         11: data = 3;
44         12: data = 70;
45         13: data = 33;
46         14: data = 6;
47         default: data = 14;
48     endcase
49 end
50
51 endmodule

```