# Lab 3: Advanced Topics Using VHDL

*University of Guelph*
ENGG*3050 Winter 2016
Professor Shawki Areibi

Graham Thoms 0795111
Vithursan Thangarasa 0821795

Group 6

# 1 Introduction

This lab required the use of VHDL to be used as tool for design entry and design the following circuits:

- 3-bit full adder using different architecture configuration specification for each bit
- N-bit counter making use of generics

The circuits were then tested and debugged by mapping on to the NEXYS3 Spartan-6 FPGA board, and performing simulations using Xilinx ISE Simulator.

# 2 Summary of Circuit Designs

This section outlines the preliminary work, implementation approaches, and simulation results of the respective circuits.

## 2.1 3-bit Full Adder with Architecture Configuration Specification

### 2.1.1 Truth Table for Specified Full Adder

The table presented in Table 1 is the truth table for the required full adder. The block diagram for the 1-bit full and 3-bit full adder is depicted in **Figure 1**.

**Table 1** Shows the truth table of the specified ALU.

| Cin | X | Y | Sum | Cout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

### 2.1.2 1-Bit and 3-Bit Full Adder

Inputs X and Y are sent into the full adder along with the carry in bit to produce the sum and carry-out bits. In order to create a 3-bit full adder three 1-bit full adders were cascaded to produce a 3-bit output. Each bit of the 3-bit full adder used a different architecture configuration that can be seen in the VHDL Implementation found in **Appendix A**.

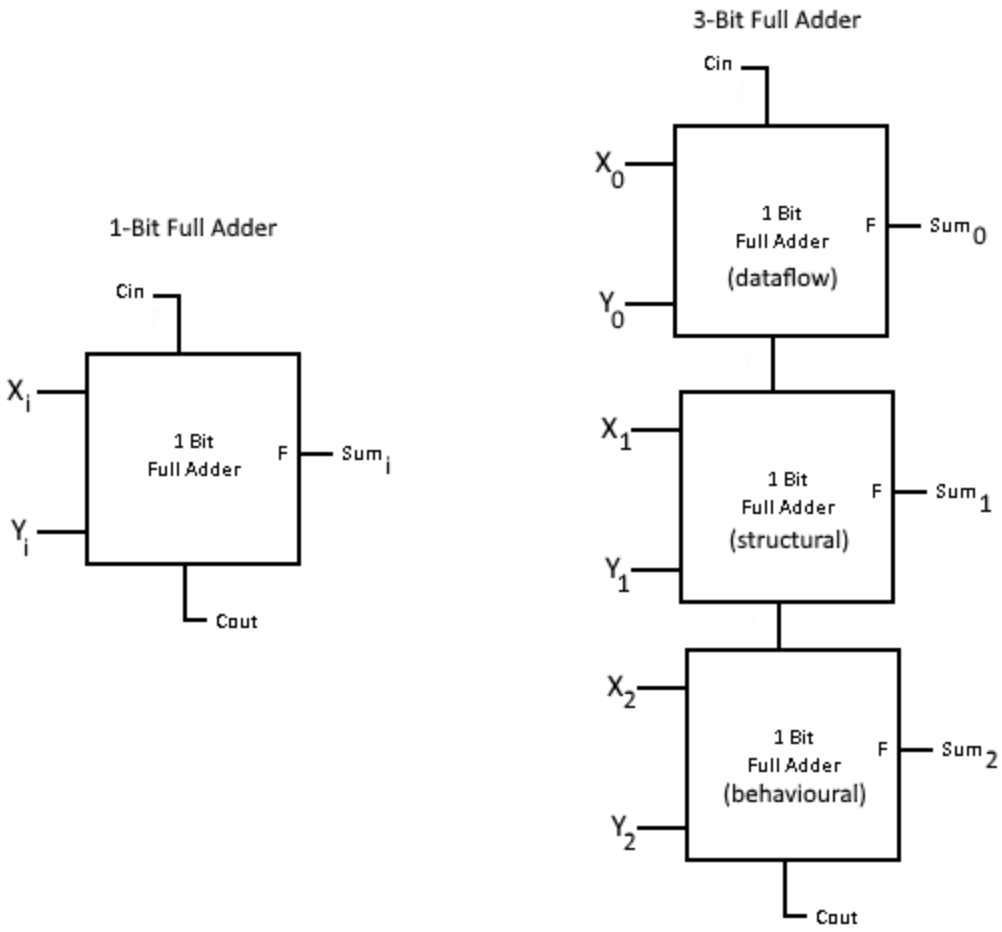


**Figure 1** Shows the implementation of the 1-bit and 3-bit full adder with architecture specification configuration.

### 2.1.3 Testbench for 3-Bit Full Adder using Architecture Specification Configurations

In **Figure 2** input A was held at a fixed number of 6 with changing values of B and Ci in order to test the sum and carry out bits, output = {Co, S(2), S(1), S(0)}. Simulations were proven to give correct output, for example when A = 6 , B = 5, Ci = 0 the output was {Co, S(2), S(1), S(0)} = {1010} = 11. Implementation of the test bench can be found in Appendix B.

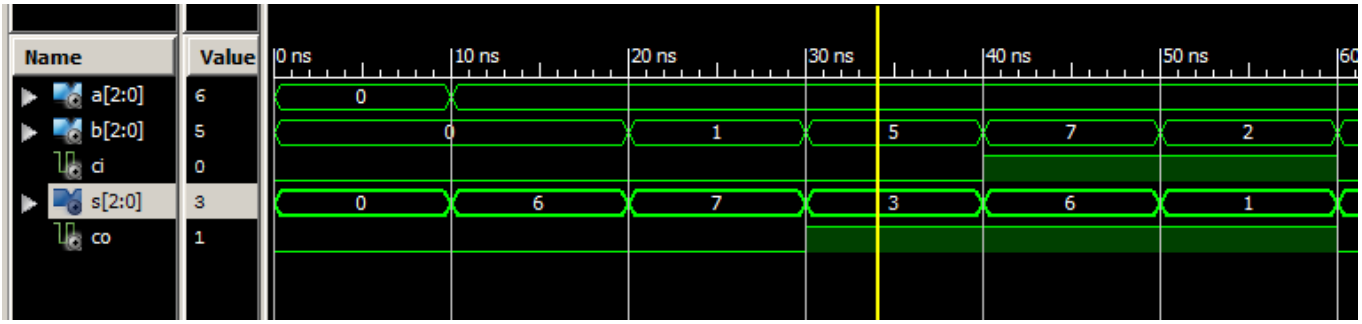


**Figure 2** Shows the logic waveform of the 3-bit Full Adder.

Three switches each were used for inputs A and B to represent bits with a push button representing the carry in bit. LEDs were used to indicate final sum and carry out bit on the NEXYS3 board. UCF file is shown below in **Figure 3**.

FullAdder_UCF.ucf

```
1   # input A(0,1,2)
2   NET A(0) LOC=T10;
3   NET A(1) LOC=T9;
4   NET A(2) LOC=V9;
5
6   # input B(0,1,2)
7   NET B(0) LOC=N8;
8   NET B(1) LOC=U8;
9   NET B(2) LOC=V8;
10
11  # input carry in
12  NET Ci LOC=D9;
13
14  # output sum(0,1,2)
15  NET S(0) LOC=U16;
16  NET S(1) LOC=V16;
17  NET S(2) LOC=U15;
18
19  # output carry out
20  NET Co LOC=V15;
```

**Figure 3** Shows the UCF for the 3-bit Full Adder.

## 2.2 N-Bit Counter

The behavioural and testbench VHDL code can be found in **Appendix C** and **D**.

### 2.2.1 Implementation Approach to the N-Bit Counter

A counter that follows the binary number sequence is called a binary counter. In VHDL language it is possible to describe a variable-size counter by using a generic declaration. The switch (“T10”) was used to simulate the clock, “rising edge” and “falling edge”. When a clock event occurs, a value of “1” is added to the n-bit “count” vector.The push button (“D9”) was used to reset and clear all bits to “0”. Depending on the the value of “n”, the LEDs were used to show the count value in binary on the NEXYS3 board. For demonstration purposes, the maximum value of the counter is 8-bits since there are a total of 8 LEDs. The UCF file is shown below in **Figure 4**.

**nBitCounter_UCF.ucf**

```
1    NET clk LOC=T10;
2
3    NET clr LOC=D9;
4
5    NET q[0] LOC=U16;
6    NET q[1] LOC=V16;
7    NET q[2] LOC=U15;
8    NET q[3] LOC=V15;
9    NET q[4] LOC=M11;
10   NET q[5] LOC=N11;
11   NET q[6] LOC=R11;
12   NET q[7] LOC=T11;
```

**Figure 4** Shows the UCF file for the n-bit Counter.

### 2.2.2 Testbench for N-Bit Counter

In **Figure 5** the testbench simulation for the counter was performed for a 4-bit counter. As seen below, when the clock is on a rising edge, the counter counts from “0000” to “1111” increasing by “0001”. Thus, the output vector “q” presents this binary number sequence accordingly.

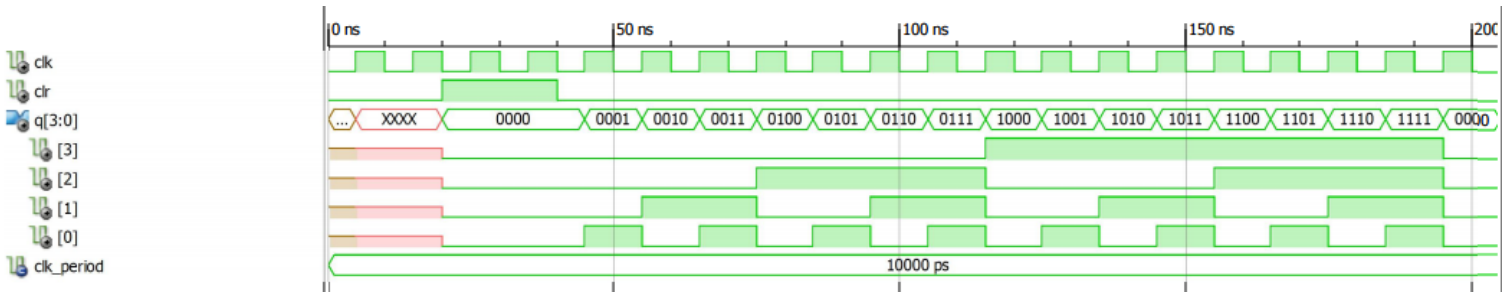


**Figure 5** Shows the logic waveform of the n-bit Counter.

### 2.2.3 Resources for 4-Bit, 8-Bit, 16-Bit Counter

The resources used for 4-Bit, 8-Bit, and 16-Bit counters are presented in **Figures 6**, **7**, and **8** respectively. It can be seen that the number of LUTs and Slice Registers required is proportional to the "n" value of the n-bit counter. The number of IOBs used is 2 more than the number of Slice Registers.

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 4 | 18224 | 0% |
| Number of Slice LUTs | 4 | 9112 | 0% |
| Number of fully used LUT-FF pairs | 0 | 8 | 0% |
| Number of bonded IOBs | 6 | 232 | 2% |
| Number of BUFG/BUFGCTRLs | 1 | 16 | 6% |

**Figure 6** Resources used for 4-Bit Counter.

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 8 | 18224 | 0% |
| Number of Slice LUTs | 8 | 9112 | 0% |
| Number of fully used LUT-FF pairs | 0 | 16 | 0% |
| Number of bonded IOBs | 10 | 232 | 4% |
| Number of BUFG/BUFGCTRLs | 1 | 16 | 6% |

**Figure 7** Resources used for 8-Bit Counter.

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 16 | 18224 | 0% |
| Number of Slice LUTs | 16 | 9112 | 0% |
| Number of fully used LUT-FF pairs | 16 | 16 | 100% |
| Number of bonded IOBs | 18 | 232 | 7% |
| Number of BUFG/BUFGCTRLs | 1 | 16 | 6% |

**Figure 8** Resources used for 16-Bit counter.

## 3 Conclusion

In the final analysis, this lab helped develop a stronger understanding of using configuration specification to change the architecture of a single entity when designing the 3-Bit Full Adder. Also, it helped show how the generic declaration can be used to create generic circuits, as presented in the implementation of the n-bit counter.

## 4 Recommendations

In the future, students can be asked to implement a more advanced n-bit circuit using the generic declaration. As a result, this will help gain more experience in advanced VHDL topics, and build further on ENGG*2410.

## Appendix A

FullAdder1bit.vhd — Mon Feb 22 23:11:00 2016

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer: Graham Thoms
4   --
5   -- Create Date:    19:44:35 02/20/2016
6   -- Design Name:
7   -- Module Name:    FullAdder1bit - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32
33  ----------------------------------------------------------------------
34  --                      1 BIT FULL ADDER
35  ----------------------------------------------------------------------
36
37  entity FullAdder1bit is
38      Port ( a   : in   STD_LOGIC;     -- bit a
39             b   : in   STD_LOGIC;     -- bit b
40             ci  : in   STD_LOGIC;     -- bit carry in
41             s   : out  STD_LOGIC;     -- bit sum = a + b + ci
42             co  : out  STD_LOGIC);    -- bit carry out
43  end FullAdder1bit;
44
45  ----------------- data flow ---------------------------
46
47  architecture FA_dataflow of FullAdder1bit is
48
49     -- dataflow of fulladder
50
51  begin
52
53     -- logic gates for full adder design
54
55     s <= ((a xor b) xor ci);                  -- sum
56     co <= ((a and b) or ((a xor b) and ci));  -- carry out
57
```

```
58  end FA_dataflow;
59
60  ----------------- end data flow ----------------------
61
62  ----------------- structural -------------------------
63
64  architecture FA_struct of FullAdder1bit is
65
66     -- components needed for building circuit
67
68     component xor_2 is
69         Port ( x : in   STD_LOGIC;
70                y : in   STD_LOGIC;
71                g : out  STD_LOGIC);
72     end component;
73
74     component and_2 is
75         Port ( x : in   STD_LOGIC;
76                y : in   STD LOGIC;
77                g : out  STD_LOGIC);
78     end component;
79
80     component or_2 is
81         Port ( x : in   STD LOGIC;
82                y : in   STD_LOGIC;
83                g : out  STD_LOGIC);
84     end component;
85
86     -- intermediate wires
87
88     signal wireABxor, wireABand, wireABCand:STD_LOGIC;
89
90     begin
91
92     -- porting of individual components to make full adder
93
94        U0: xor_2 port map(a,b,wireABxor);
95        U1: and_2 port map(a,b,wireABand);
96        U2: xor_2 port map(wireABxor,ci,s);
97        U3: and_2 port map(wireABxor,ci,wireABCand);
98        U4: or_2  port map(wireABCand,wireABand,co);
99
100 end FA struct;
101
102 ----------------- end structural ---------------------
103
104 ----------------- behavioural ------------------------
105
106 architecture FA_Behav of FullAdder1bit is
107
108 begin
109
110    FA: process(a,b,ci)
111    begin
112
113    -- multiplexer logic for full adder
114
```

```
115             if(a = '0' and b = '0' and Ci = '0')then s <= '0'; co <= '0';
116           elsif(a = '0' and b = '0' and Ci = '1')then s <= '1'; co <= '0';
117           elsif(a = '0' and b = '1' and Ci = '0')then s <= '1'; co <= '0';
118           elsif(a = '0' and b = '1' and Ci = '1')then s <= '0'; co <= '1';
119           elsif(a = '1' and b = '0' and Ci = '0')then s <= '1'; co <= '0';
120           elsif(a = '1' and b = '0' and Ci = '1')then s <= '0'; co <= '1';
121           elsif(a = '1' and b = '1' and Ci = '0')then s <= '0'; co <= '1';
122           else  s <= '1'; co <= '1';
123           end if;
124
125       end process FA;
126
127   end FA_Behav;
128
129   ----------------- end behavioural --------------------
130
131   ----------------------------------------------------------------------
132   --                        END 1 BIT FULL ADDER
133   ----------------------------------------------------------------------
134   ----------------------------------------------------------------------
135   --                        XOR2
136   ----------------------------------------------------------------------
137   library IEEE;
138   use IEEE.STD_LOGIC_1164.ALL;
139
140   -- two input XOR gate
141
142   entity xor_2 is
143       Port ( x : in   STD_LOGIC;       -- input x
144              y : in   STD_LOGIC;       -- input y
145              g : out  STD_LOGIC);      -- output g = x and y
146   end xor_2;
147
148   architecture structural of xor_2 is
149
150   begin
151      g <= x xor y;  -- output
152
153   end structural;
154   ----------------------------------------------------------------------
155   --                        END XOR2
156   ----------------------------------------------------------------------
157   ----------------------------------------------------------------------
158   --                        AND2
159   ----------------------------------------------------------------------
160   library IEEE;
161   use IEEE.STD_LOGIC_1164.ALL;
162
163   -- two input AND gate
164
165   entity and_2 is
166       Port ( x : in   STD_LOGIC;       -- input x
167              y : in   STD_LOGIC;       -- input y
168              g : out  STD_LOGIC);      -- output g = x and y
169   end and_2;
170
171   architecture structural of and_2 is
```

```
172
173   begin
174      g <= x and y;  -- output
175
176   end structural;
177   ----------------------------------------------------------------------
178   --                        END AND2
179   ----------------------------------------------------------------------
180   ----------------------------------------------------------------------
181   --                        OR2
182   ----------------------------------------------------------------------
183   library IEEE;
184   use IEEE.STD_LOGIC_1164.ALL;
185
186   -- two input OR gate
187
188   entity or_2 is
189       Port ( x : in   STD_LOGIC;      -- input x
190              y : in   STD LOGIC;      -- input y
191              g : out  STD_LOGIC);     -- output g = x or y
192   end or_2;
193
194   architecture structural of or_2 is
195
196   begin
197      g <= x or y;   -- output
198
199   end structural;
200   ----------------------------------------------------------------------
201   --                        END OR2
202   ----------------------------------------------------------------------
203
204
```

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer: Graham Thoms
4  --
5  -- Create Date:    17:44:59 02/20/2016
6  -- Design Name:
7  -- Module Name:    FullAdder_VHDL - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use WORK.all;
23
24 -- Uncomment the following library declaration if using
25 -- arithmetic functions with Signed or Unsigned values
26 --use IEEE.NUMERIC_STD.ALL;
27
28 -- Uncomment the following library declaration if instantiating
29 -- any Xilinx primitives in this code.
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 ----------------------------------------------------------------------
34 --                      3 BIT FULL ADDER
35 ----------------------------------------------------------------------
36
37 entity FullAdder_VHDL is
38     Port ( A   : in   STD_LOGIC_VECTOR(2 downto 0);      -- input bits A1, A2, A3
39            B   : in   STD_LOGIC_VECTOR (2 downto 0);     -- input bits B1, B2, B3
40            Ci  : in   STD_LOGIC;                         -- Carry in bit
41            S   : out  STD_LOGIC_VECTOR (2 downto 0);     -- output sum S(i) = A(i)+B(i)
42            Co  : out  STD_LOGIC);                        -- carry out bit
43 end FullAdder VHDL;
44
45 architecture Behavioral of FullAdder_VHDL is
46
47    -- uses three 1 bit full adders cascaded together
48
49    component FullAdder1bit is
50        Port ( a   : in   STD_LOGIC;
51               b   : in   STD_LOGIC;
52               ci  : in   STD_LOGIC;
53               s   : out  STD_LOGIC;
54               co  : out  STD_LOGIC);
55    end component;
56
57    -- each bit has a different architecture configuration
```

```
58
59      for B0: FullAdder1bit use entity WORK.FullAdder1bit (FA_dataflow);
60      for B1: FullAdder1bit use entity WORK.FullAdder1bit (FA_struct);
61      for B2: FullAdder1bit use entity WORK.FullAdder1bit (FA_Behav);
62
63      signal cout0,cout1,cout2: STD_LOGIC;
64
65  begin
66
67      -- three 1-bit full adders
68
69      B0: FullAdder1bit port map (A(0),B(0),Ci,S(0),cout0);
70      B1: FullAdder1bit port map (A(1),B(1),cout0,S(1),cout1);
71      B2: FullAdder1bit port map (A(2),B(2),cout1,S(2),cout2);
72
73      -- carry out bit
74
75      Co <= cout2;
76
77  end Behavioral;
78
79  ----------------------------------------------------------------------
80  --                        END 3 BIT FULL ADDER
81  ----------------------------------------------------------------------
82
83
```

## Appendix B

FullAdderVHDL_testbench.vhd Mon Feb 22 23:46:17 2016

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:   Graham Thoms
4   --
5   -- Create Date:   23:22:53 02/22/2016
6   -- Design Name:
7   -- Module Name:   H:/ENGG3050/FullAdderVHDL/FullAdderVHDL_testbench.vhd
8   -- Project Name:  FullAdderVHDL
9   -- Target Device:
10  -- Tool versions:
11  -- Description:
12  --
13  -- VHDL Test Bench Created by ISE for module: FullAdder_VHDL
14  --
15  -- Dependencies:
16  --
17  -- Revision:
18  -- Revision 0.01 - File Created
19  -- Additional Comments:
20  --
21  -- Notes:
22  -- This testbench has been automatically generated using types std_logic and
23  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24  -- that these types always be used for the top-level I/O of a design in order
25  -- to guarantee that the testbench will bind correctly to the post-implementation
26  -- simulation model.
27  ----------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY FullAdderVHDL_testbench IS
36  END FullAdderVHDL_testbench;
37
38  ARCHITECTURE behavior OF FullAdderVHDL_testbench IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT FullAdder_VHDL
43      PORT(
44           A : IN  std_logic_vector(2 downto 0);
45           B : IN  std_logic_vector(2 downto 0);
46           Ci : IN  std_logic;
47           S : OUT  std_logic_vector(2 downto 0);
48           Co : OUT  std_logic
49          );
50      END COMPONENT;
51
52
53      --Inputs
54      signal A : std_logic_vector(2 downto 0) := (others => '0');
55      signal B : std_logic_vector(2 downto 0) := (others => '0');
56      signal Ci : std_logic := '0';
57
```

```
58      --Outputs
59      signal S : std_logic_vector(2 downto 0);
60      signal Co : std_logic;
61      -- No clocks detected in port list. Replace <clock> below with
62      -- appropriate port name
63
64      constant period : time := 10 ns;
65
66    BEGIN
67
68      -- Instantiate the Unit Under Test (UUT)
69      uut: FullAdder_VHDL PORT MAP (
70             A => A,
71             B => B,
72             Ci => Ci,
73             S => S,
74             Co => Co
75           );
76
77      -- Stimulus process
78      stim_proc: process
79      begin
80         -- hold reset state for 100 ns.
81         wait for period;
82
83         -- keeping A = 6 and changing B and Ci
84
85         A(0) <= '0'; -- 6
86         A(1) <= '1';
87         A(2) <= '1';
88
89         B(0) <= '0';
90         B(1) <= '0';
91         B(2) <= '0';
92
93         Ci <= '0';
94
95         wait for period;
96
97         -- 6 + 1 = 7
98
99         B(0) <= '1'; -- 1
100        B(1) <= '0';
101        B(2) <= '0';
102
103        Ci <= '0';
104
105        wait for period;
106
107        -- 6 + 5 = 11
108
109        B(0) <= '1'; -- 5
110        B(1) <= '0';
111        B(2) <= '1';
112
113        Ci <= '0';
114
```

```
115         wait for period;
116
117         -- 6 + 8 = 14
118
119         B(0) <= '1'; -- 7
120         B(1) <= '1';
121         B(2) <= '1';
122
123         Ci <= '1';
124
125         wait for period;
126
127         -- 6 + 3 = 9
128
129         B(0) <= '0'; -- 2
130         B(1) <= '1';
131         B(2) <= '0';
132
133         Ci <= '1';
134
135         wait for period;
136
137         wait;
138      end process;
139
140   END;
141
```

## Appendix C

nBitCounter.vhd — Mon Feb 22 12:49:13 2016

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    12:02:34 02/22/2016
6   -- Design Name:
7   -- Module Name:    nBitCounter - Behavioral
8   -- Project Name:
9   -- Target Devices:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library ieee ;
21  use ieee.std_logic_1164.all;
22  use ieee.std_logic_unsigned.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity nBitCounter is
34
35  generic(n: integer :=4);
36  port( clk:  in std_logic;
37        clr:  in std_logic;
38        q : out std_logic_vector(n-1 downto 0)
39  );
40  end nBitCounter;
41
42  architecture Behavioral of nBitCounter is
43
44      signal count: std_logic_vector(n-1 downto 0); -- create signal for the output
45
46  begin
47     -- behaviour describes the counter
48     process(clk, clr)
49     begin
50        if clr = '1' then
51           count <= (others => '0');
52        elsif clk'event and clk = '1' then
53           count <= count + 1;
54        end if;
55     end process;
56     -- concurrent assignment statement
57     q <= count;
```

nBitCounter.vhd — Mon Feb 22 12:49:14 2016

```
58  end Behavioral;
59
60
```

## Appendix D

nBitCounter_TB.vhd — Mon Feb 22 12:48:00 2016

```
1   --------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:   12:34:37 02/22/2016
6   -- Design Name:
7   -- Module Name:   C:/Users/vthangar/Documents/counter/nBitCounter/nBitCounter_TB.vhd
8   -- Project Name:  nBitCounter
9   -- Target Device:
10  -- Tool versions:
11  -- Description:
12  --
13  -- VHDL Test Bench Created by ISE for module: nBitCounter
14  --
15  -- Dependencies:
16  --
17  -- Revision:
18  -- Revision 0.01 - File Created
19  -- Additional Comments:
20  --
21  -- Notes:
22  -- This testbench has been automatically generated using types std_logic and
23  -- std_logic_vector for the ports of the unit under test.  Xilinx recommends
24  -- that these types always be used for the top-level I/O of a design in order
25  -- to guarantee that the testbench will bind correctly to the post-implementation
26  -- simulation model.
27  --------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY nBitCounter_TB IS
36  END nBitCounter_TB;
37
38  ARCHITECTURE behavior OF nBitCounter_TB IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT nBitCounter
43      PORT(
44           clk : IN  std_logic;
45           clr : IN  std_logic;
46           q : OUT  std_logic_vector(3 downto 0)
47          );
48      END COMPONENT;
49
50
51     --Inputs
52     signal clk : std_logic := '0';
53     signal clr : std_logic := '0';
54
55     --Outputs
56     signal q : std_logic_vector(3 downto 0);
57
```

```
58       -- Clock period definitions
59       constant clk_period : time := 10 ns;
60
61    BEGIN
62
63       -- Instantiate the Unit Under Test (UUT)
64       uut: nBitCounter PORT MAP (
65              clk => clk,
66              clr => clr,
67              q => q
68            );
69
70       -- Clock process definitions
71       clk_process :process
72       begin
73          clk <= '0';
74          wait for clk_period/2;
75          clk <= '1';
76          wait for clk_period/2;
77       end process;
78
79
80       -- Stimulus process
81       stim_proc: process
82       begin
83          wait for clk_period*2;
84          clr <= '1';
85          wait for clk_period*2;
86          clr <= '0';
87
88          wait;
89       end process;
90
91    END;
92
```