

Sam Harries Assignment 1 - Final Version

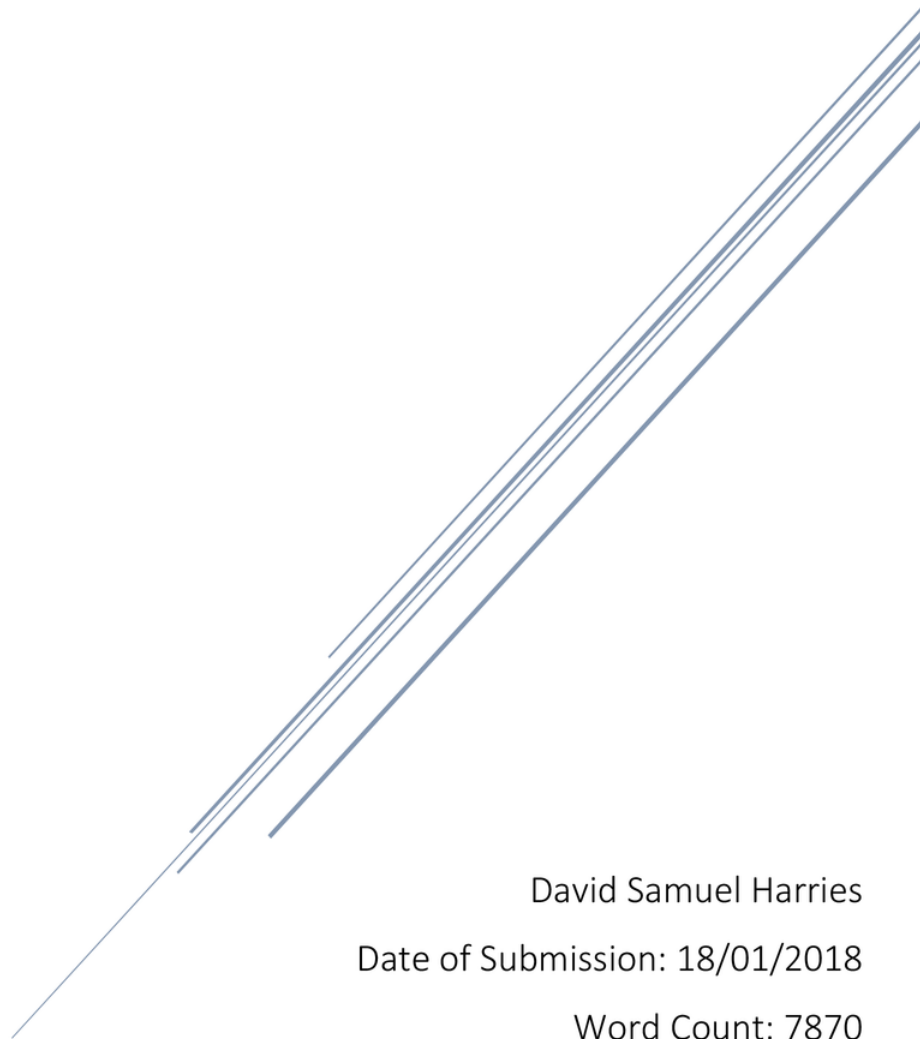
by David Harries

FILE	7309_DAVID_HARRIES_SAM_HARRIES_ASSIGNMENT_1_-_FINAL_VERSION_155114_477819469.DOCX (5.34 M)		
TIME SUBMITTED	18-JAN-2018 10:01AM (UTC+0000)	WORD COUNT	7858
SUBMISSION ID	81988625	CHARACTER COUNT	46116

HND Electrical and Electronic Engineering

PSDE5027: Virtual Instrumentation and Control

Assignment 1: Binary / BCD Conversion in VHDL



David Samuel Harries

Date of Submission: 18/01/2018

Word Count: 7870

TABLE OF CONTENTS

1	– Introduction	4
2	Research.....	6
2.1	Structure of a Typical VHDL File	6
2.1.1	Entity	6
2.1.2	Architecture	6
2.1.3	Package	10
2.1.4	Configuration	10
2.1.5	Library	10
2.2	Binary to BCD Conversion	10
2.3	IP Block Diagrams in Vivado Suite	12
3	Design of Assignment Requirements	13
3.1	6 Bit BCD Converter	13
3.2	8 Bit Binary to BCD Converter	17
4	Analysis of Xilinx Vivado Simulated Results	21
4.1	6 Bit Binary to BCD Converter Simulated Results	21
4.2	8 Bit Binary to BCD Converter Simulated Results	22
5	Analysis of Nexys4 FPGA Practical Results	25
6	Adapting Design to Suit New Purposes or Applications	27
6.1	5KHz Clock from Internal 100MHz Clock	27
6.2	Mux_Demux.....	30
6.3	Seven Segment Display	34
7	Analysis of Xilinx Vivado Simulated Results	39
7.1	5KHz Clock Simulation Test	39
7.2	Mux_Demux Simulation Test	40
7.3	BCD_7seg Simulation Test	41
8	Analysis of Nexys 4 DDR FPGA Practical Results.....	44
9	Conclusion.....	48
10	References	49
11	Appendices.....	50

Table of Figures

Figure 1-1	4
Figure 1-2	5
Figure 2-1	6

Figure 2-2	7
Figure 2-3	8
Figure 2-4	9
Figure 2-5	10
Figure 2-6	10
Figure 2-7	10
Figure 2-8	11
Figure 2-9	11
Figure 2-10	11
Figure 3-1	13
Figure 3-2	13
Figure 3-3	15
Figure 3-4	16
Figure 3-5	17
Figure 3-6	18
Figure 3-7	19
Figure 3-8	19
Figure 4-1	21
Figure 4-2	21
Figure 4-3	22
Figure 4-4	22
Figure 4-5	23
Figure 4-6	23
Figure 5-1	25
Figure 5-2	25
Figure 5-3	26
Figure 5-4	26
Figure 6-1	27
Figure 6-2	27
Figure 6-3	28
Figure 6-4	29
Figure 6-5	29
Figure 6-6	30
Figure 6-7	31
Figure 6-8	33
Figure 6-9	33
Figure 6-10	34
Figure 6-11	35
Figure 6-12	37
Figure 7-1	39
Figure 7-2	39
Figure 7-3	40
Figure 7-4	40
Figure 7-5	41
Figure 7-6	41
Figure 7-7	42
Figure 7-8	42
Figure 8-1	44

Figure 8-2	45
Figure 8-3	45
Figure 8-4	46
Figure 8-5	46
Figure 8-6	46
Figure 8-7	47

1 – INTRODUCTION

The aims of this assignment are to design and create a collection of VHDL files that when combined will create an 8 bit Binary to BCD converter that will display a value ranging from 0 to 255 on a Nexys 4 DDR FPGA. The following steps will be carried out in order to fulfil this aim:

1. Creating a 6 bit Binary to BCD Converter.
2. Alter the design to create an 8 bit Binary to BCD Converter.
3. Test the converter on the Nexys 4 DDR FPGA using IP Block Diagrams.
4. Adapt the design so that the BCD value is displayed on 3*7 segment displays when the same Binary value is input using switches.

The specification of the assignment states that the datasheet for the 74185A chip should be analysed. This chip is a 6 bit Binary to BCD Converter and through analysis of the datasheet; a VHDL file can be made. This can then be extended by cascading two 6 bit converters to create an 8 bit converter. This will allow the indication of a BCD value ranging from 0 – 255 from 8 input switches located on the Nexys board. This value can then be displayed on a series of LEDs. The inputs and outputs can be seen below.

Inputs: SW0 – SW7 representing the 8 bit Binary input.

Outputs: LED3 – LED0 representing units where LED3 is the most significant bit (MSB).

LED7 – LED4 representing the tens where LED7 is the MSB.

LED9 – LED8 representing the hundreds where LED9 is the MSB.

The specification also provides a block diagram indicating how the 8 bit BCD converter must be adapted to display the BCD value on the 3*7 segment displays located on the Nexys board. This can be seen below in **Figure 1-1**.

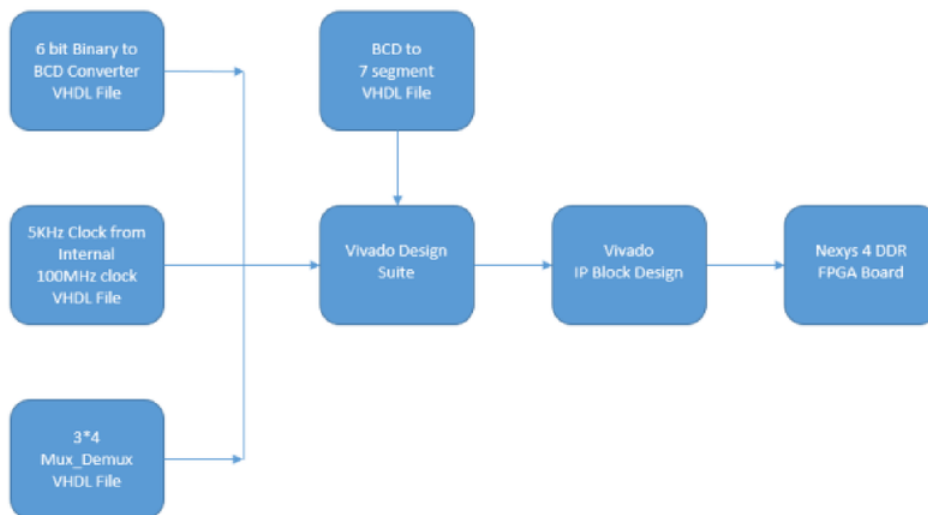
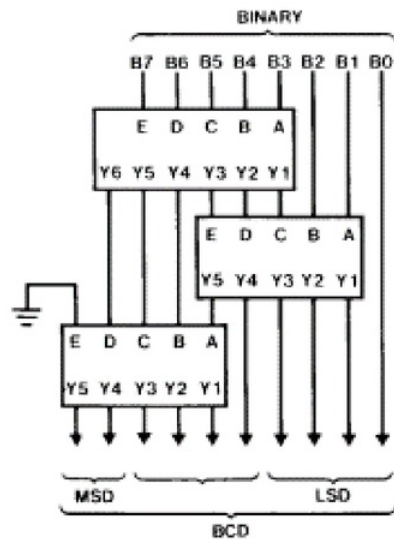


Figure 1-1

From the block diagram it can be seen that multiple VHDL files must be created in order to fulfil the specification. These include:

- 6 bit Binary to BCD Converter.
- 5kHz Clock from Internal 100MHz clock.
- 3*4 Mux_Demux.
- BCD to 7 Segment.

The design and testing of these can be seen in further sections within the report. The following 8 bit diagram can be found from the datasheet for the DM74185. This configuration shows that three 6 bit binary to BCD converters can be used to create an 8 bit binary to BCD converter. This is the configuration that will be used in the design later on.



Good detail...

Figure 1-2

2 RESEARCH

The following topics will be researched within this section of the report:

- Structure of a typical VHDL file.
- Binary to BCD Conversion.
- IP Block Design in the Vivado Design Suite.

2.1 STRUCTURE OF A TYPICAL VHDL FILE

The typical structure of a VHDL file consists of five groups; Entity, Architecture, Package, configuration and Library.

2.1.1 Entity

The entity of a VHDL file can be considered the interface to the outside world. In an entity declaration, the name of the VHDL file along with any inputs or outputs are stated. A typical entity declaration can be seen in **Figure 2-1**.

```
entity NAME_OF_ENTITY is [ generic generic_declarations];  
  
port (signal_names: mode type;  
  
      signal_names: mode type;  
  
      :  
  
      signal_names: mode type);  
  
end [NAME_OF_ENTITY];
```

Figure 2-1

An entity declaration begins with the keyword *entity* which is then followed by the entity's name and the keyword *is*. Following this is the declaration of any input or output ports that the VHDL file will have. This is done using the keyword *port*. To end the entity the keyword *end* is used which is then followed by the name of the entity within square brackets.

[Good detail](#)

2.1.2 Architecture

Whereas the entity acts as an interface to the outside world for the VHDL file, the architecture is the internal workings of the VHDL file. The architecture body uses the stated inputs and outputs from the entity to determine how the circuit actually operates. This can be done using a behavioural model or a structural model. A typical architecture body can be seen in **Figure 2-2**.

```
architecture architecture_name of NAME_OF_ENTITY is
```

```
    — Declarations
```

```
    — components declarations
```

```
    — signal declarations
```

```
    — constant declarations
```

```
    — function declarations
```

```
    — procedure declarations
```

```
    — type declarations
```

```
;
```

```
begin
```

```
    — Statements
```

```
;
```

```
end architecture_name;
```

Figure 2-2

In a behavioural model the architecture body is described at a behavioural level. This means the VHDL code describes the behaviour of the components. An example of this can be seen in **Figure 2-3** which shows a behavioural description of a two-input AND gate.

```
entity AND2 is

    port (in1, in2: in std_logic;

        out1: out std_logic);

end AND2;


architecture behavioral_2 of AND2 is

begin

    out1 <= in1 and in2;

end behavioral_2;
```

Figure 2-3

This example shows an entity with two inputs (in1 and in2) and one output (out1). In the architecture body, the behaviour of the circuit is shown as out1 will be active if in1 AND in2 are both active. This is the operation of an AND gate. The architecture body is ended with the keyword *end*.

In a structural model, the circuit is described by specifying what gates are used and how they are all connected to each other. An example of this can be seen in **Figure 2-4**. It can be seen that in the declarations, the components are all defined. This is then followed by a declaration of signals. Finally, following the keyword *begin*, component instantiations statements are declared using the keyword *port map*. The architectural body is then ended using the keyword *end* just like before.

architecture structural of BUZZER is

— Declarations

component AND2

port (in1, in2: in std_logic;

out1: out std_logic);

end component;

component OR2

port (in1, in2: in std_logic;

out1: out std_logic);

end component;

component NOT1

port (in1: in std_logic;

out1: out std_logic);

end component;

— declaration of signals used to interconnect gates

signal DOOR_NOT, SBELT_NOT, B1, B2: std_logic;

begin

— Component instantiations statements

U0: NOT1 port map (DOOR, DOOR_NOT);

U1: NOT1 port map (SBELT, SBELT_NOT);

U2: AND2 port map (IGNITION, DOOR_NOT, B1);

U3: AND2 port map (IGNITION, SBELT_NOT, B2);

U4: OR2 port map (B1, B2, WARNING);

end structural;

Figure 2-4

2.1.3 Package

A VHDL package is a file which contains commonly used declarations of components, signals, functions, objects or data types which can then be used in different VHDL models.

2.1.4 Configuration

Configurations allow a designer to alter and adapt VHDL files with different variations of a design by selecting different configurations. This is done by linking component instances of a structure design to entity-architecture pairs.

2.1.5 Library

A library is a place where the compiler stores information that can be used in a design project. An example of this would be std_logic. Standard logic or std_logic is defined within the package ieee.std_logic_1164. This package is then stored within the ieee library. At the start of VHDL files in which std_logic is required the ieee library can be declared through the use of the keyword *library*. This can be seen in **Figure 2-5**. The *.all* keyword indicates for the entire ieee.std_logic_1164 package to be used.

```
library ieee;  
  
use ieee.std_logic_1164.all;
```

Figure 2-5

Good research but not referenced?

2.2 BINARY TO BCD CONVERSION

The purpose of Binary to BCD Conversion is to separate a binary number into parts each representing a digit of a decimal number. This can be seen in **Figure 2-6**.

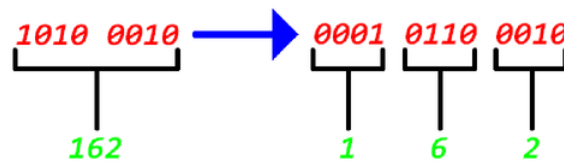


Figure 2-6

Excellent!

This conversion can be done in a few ways. The first of which is manually converting the binary number to a decimal and then converting each digit of the decimal number back into binary. As an example the binary number 11101_2 converted into BCD would be done as follows:

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Figure 2-7

This means that 11101_2 in decimal is 29. The next step is to convert the 2 and the 9 back into binary.

Step	Decimal Number	Conversion
Step 1	29_{10}	$0010_2\ 1001_2$
Step 2	29_{10}	00101001_{BCD}

Figure 2-8

The final BCD value is 00101001_{BCD} . Another method of conversion is through the use of an algorithm. This algorithm has four steps:

1. If any column (100's, 10's, 1's, etc.) is 5 or greater, add 3 to that column.
2. Shift all #'s to the left 1 position.
3. If 8 shifts have been performed, it's done! Evaluate each column for the BCD values.
4. Go to step 1.

This algorithm can be written in code as seen in **Figure 2-9**.

```

for(i=0; i<8; i++) {
    //check all columns for >= 5
    for each column {
        if (column >= 5)
            column += 3;

        //shift all binary digits left 1
        Hundreds <<= 1;
        Hundreds[0] = Tens[3];
        Tens <<= 1;
        Tens[0] = Ones[3];
        Ones <<= 1;
        Ones[0] = Binary[7];
        Binary <<= 1;
    }
}

```

Figure 2-9

The result of this algorithm can be seen below in **Figure 2-10**.

100's	10's	1's	Binary	Operation
			1010 0010	← 162
		1	010 0010	<< #1
		10	10 0010	<< #2
		101	0 0010	<< #3
		1000		add 3
	1	0000	0010	<< #4
	10	0000	010	<< #5
	100	0000	10	<< #6
	1000	0001	0	<< #7
	1011			add 3
1	0110	0010		<< #8

↑ ↑ ↑ Very good detail
1 6 2

Figure 2-10

2.3 IP BLOCK DIAGRAMS IN VIVADO SUITE

When a VHDL file has been fully designed and tested, it can be saved as an IP block diagram. This means that the file is given a diagram which resembles a block which has input and output ports. These ports are the same as the ones stated in the entity of the VHDL file. When saved as an IP block diagram, the file is put into an IP repository which allows the component created in the VHDL to be used in other designs. When a lot of designs have been made and saved as IPs, they can all be used in a fresh design by simply importing the IP block diagrams and connecting them in the configuration the designer desires. This is useful as it saves time re-designing components, and allows reuse of previously designed ones. Below shows an example of IP block diagrams in use.

*Where is the diagram? More detail
required*

3 DESIGN OF ASSIGNMENT REQUIREMENTS

3.1 6 BIT BCD CONVERTER

This section will detail the design of the 6 bit Binary to BCD converter. This is the first VHDL file that is required to fulfil the assignment briefs requirements. Once the 6 bit converter has been designed it will then be simulated in Vivado. The first stage of the design process after setting up the Vivado file is to create a design source. The VHDL code is written within the design source which includes the entity and architecture of the component. Sections of the VHDL design source can be seen in **Figure 3-1** and **Figure 3-2**.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Ass_p1 is
4  Port ( A,B,C,D,E : in STD_LOGIC;
5        Y1,Y2,Y3,Y4,Y5,Y6 : out STD_LOGIC);
6  end Ass_p1;
7  architecture Behavioral of Ass_p1 is
8  signal Inp : std_logic_vector (4 downto 0);
9  signal Q : std_logic_vector (5 downto 0);
10 begin
11   Inp(0) <= A;
12   Inp(1) <= B;
13   Inp(2) <= C;
14   Inp(3) <= D;
15   Inp(4) <= E;
16   process (Inp)
17   begin
18     case Inp is
19     WHEN "00000" => -- 0/1
20       Q <= ("000000");
21     WHEN "00001" => -- 2/3
22       Q <= ("000001");
23     WHEN "00010" => -- 4/5
24       Q <= ("000010");
25     WHEN "00011" => -- 6/7
```

Where does
this
information
come from?

Figure 3-1

```
80   Q <= ("110000");
81   WHEN "11111" => -- 62/63
82   Q <= ("110001");
83   WHEN Others =>
84   Q <= ("000000");
85   end case;
86   end process;
87
88   Y1 <= Q(0);
89   Y2 <= Q(1);
90   Y3 <= Q(2);
91   Y4 <= Q(3);
92   Y5 <= Q(4);
93   Y6 <= Q(5);
94   end behavioral;
--
```

Figure 3-2

As shown above, the first thing that is declared within the code is the library which is IEEE. Just like in section 2.1.5, the .ALL extension is used to ensure the entire IEEE.STD_LOGIC_1163 package is used. Following this is the entity. This is done by using the keyword *entity* followed by the name of the component which is Ass_p1, which is then followed by the keyword *is*. Within the

entity, the inputs and outputs are declared. A, B, C, D and E are all inputs which will be used to input a binary code. Y1, Y2, Y3, Y4, Y5 and Y6 are all outputs which will be used to output the BCD values. The entity is then ended via the use of the keyword *end* followed by the entity name. The STD_LOGIC means that the signals used within this component can take on the following values:

- 'U': uninitialized. This signal hasn't been set yet.
- 'X': unknown. Impossible to determine this value/result.
- '0': logic 0
- '1': logic 1
- 'Z': High Impedance
- 'W': Weak signal, can't tell if it should be 0 or 1.
- 'L': Weak signal that should probably go to 0
- 'H': Weak signal that should probably go to 1
- '-': Don't care.

Following the entity is the architecture. The architecture uses the behavioural model rather than the structural model. This is declared by using the keyword *architecture* followed by Behavioural. The entity is then named afterwards to ensure that the architecture is paired with the entity. This allows the inputs and outputs that were declared within the entity to be used within the architecture. The internal signals are then declared through the use of the keyword *signal* followed by the signals name. In this case the two internal signals are Inp and Q which are both followed by STD_LOGIC_VECTOR. Inp is then set to 4 down to 0. This gives Inp 5 possible values which will be used to store the input binary values. Q is set to 5 down to 0 which gives it 6 possible values which will be used to store the output BCD values.

The architecture is then began through the keyword *begin*. The inputs A to E are then set to individual elements within Inp. The process is then declared. Within the process is the mechanism of the circuit. The code states that when Inp is set to a specific value from 0 to 63 in binary, Q will be set to the BCD equivalent. This is done by using a case statement followed by a series of When statements. The BCD values are determined using the Binary to BCD truth table shown in the appendices. Each BCD value can represent two values depending on whether the clock pulse is high or low.

The case statement is then ended along with the process. Finally, the BCD values stored within Q are assigned to the outputs Y1 to Y6. The architecture is then ended with the use of the keyword *end* followed by behavioural.

The next step was to simulate the 6 bit BCD converter. To do this a test bench file must be created. This test bench file is used to create parameters under which the VHDL design source must follow. The test bench file can be seen in **Figure 3-3** and **Figure 3-4**.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Ass_p1_tb is
4  -- Port ( );
5  end Ass_p1_tb;
6  architecture Behavioral of Ass_p1_tb is
7  -- Component Declaration for the Unit Under Test (UUT)
8  COMPONENT Ass_p1
9  Port ( A,B,C,D,E : in std_logic;
10 Y1,Y2,Y3,Y4,Y5,Y6 : out std_logic
11 );
12 END COMPONENT;
13 --Inputs
14 signal A : std_logic := '0';
15 signal B : std_logic := '0';
16 signal C : std_logic := '0';
17 signal D : std_logic := '0';
18 signal E : std_logic := '0';
19 --Outputs
20 signal Y1 : std_logic;
21 signal Y2 : std_logic;
22 signal Y3 : std_logic;
23 signal Y4 : std_logic;
24 signal Y5 : std_logic;
25 signal Y6 : std_logic;
26 --Clocks
27 constant A_period : time := 1 us;
28 constant B_period : time := 2 us;
29 constant C_period : time := 4 us;
30 constant D_period : time := 8 us;
31 constant E_period : time := 16 us;
32 BEGIN
33 -- Instantiate the Unit Under Test (UUT)
34 uut: Ass_p1 PORT MAP (
35 A => A,
36 B => B,
37 C => C,
38 D => D,
39 E => E,
40 Y1 => Y1,
41 Y2 => Y2,
42 Y3 => Y3,
43 Y4 => Y4,
44 Y5 => Y5,
45 Y6 => Y6

```

Figure 3-3

```

45     Y6 => Y6
46   );
47   -- Clock process definitions
48   A_process :process
49   begin
50     A <= '0';
51     wait for A_period/2;
52     A <= '1';
53     wait for A_period/2;
54   end process;
55   B_process :process
56   begin
57     B <= '0';
58     wait for B_period/2;
59     B <= '1';
60     wait for B_period/2;
61   end process;
62   C_process :process
63   begin
64     C <= '0';
65     wait for C_period/2;
66     C <= '1';
67     wait for C_period/2;
68   end process;
69   D_process :process
70   begin
71     D <= '0';
72     wait for D_period/2;
73     D <= '1';
74     wait for D_period/2;
75   end process;
76   E_process :process
77   begin
78     E <= '0';
79     wait for E_period/2;
80     E <= '1';
81     wait for E_period/2;
82   end process;
83   END;

```

Figure 3-4

The test bench file is begun in the same fashion as the design source by declaring the entity. However, there are no inputs or outputs to the test bench so the entity is simply ended. The purpose of this is to give the test bench a name. The architecture is then declared using the same name as the entity. The component is then stated with the name used for the component in the design source. This tells the test bench file what component needs testing. The inputs and outputs are then declared just like in the design source. The component is then ended. Following this are the signals such as inputs, outputs and clock periods. The inputs are all set to use STD_LOGIC and are defaulted to '0'. The outputs are all also set to use STD_LOGIC.

The clocks are all set to have different time periods. A_period is set to 1us, B_period is set to 2us, C_period to 4us, D_period to 8us and E_period to 16us. The significance of these various time values is the fact they have a binary relationship. This means that it will only take a period of 16us to get every result possible from the simulation. The simulation process is then began.

The first thing within the process is instantiating the unit under test (uut). In this case it's the component from the design source, so the name Ass_p1 is used. Next a port map is created. This simply assigns the ports (Inputs and Outputs) declared using the *port* keyword earlier in the architecture to the inputs and outputs used within the design source. In this case they all have the same names so A is assigned to A and so on.

Following the port map is the clock process definitions. This section of code contains a series of processes, each for a single input. The first is the clock period definition for input A. The process is began by stating the process, and using the keyword *begin*. A is then initially set to 0. The next line has a wait for command which states to wait for A_period/2. A_period was set to 1us earlier in the clocks section. Therefore, the test bench file must wait 0.5us to continue with this process. Once that 0.5us has passed, the next line can be executed which assigns A to equal 1. The same wait for command follows this line. This creates an alternating pattern for input A between high and low every 0.5us.

The next process is for input B, and is exactly the same except it uses B_period. B_period is set to 2us in the clock section of the architecture. Therefore B is set to alternate every 1us between high and low. The following three processes do exactly the same for inputs C, D and E making them alternate between high and low every 2us, 4us and 8us respectively. The test bench is then ended with the keyword *end* and is now ready to test.

Due to the time period being set to a minimum of 16us to see all possible results from the test, the testing time is set to 16us in order to get a single set of results. The VHDL file was simulated and tested for three different values. These tests can be seen in **Section 4** of the report which analyses the simulated results from this section.

3.2 8 BIT BINARY TO BCD CONVERTER

In order to fulfil the specification, an 8 bit binary to BCD converter must be made. One way of making this is to cascade a series of three 6 bit BCD converters. Firstly, the 6 bit BCD converter must be made into an IP block diagram. Once that is done it can be used as many times as needed in any further projects. Figure 3-5 shows the 6 bit BCD converter cascaded 3 times to create an 8 bit binary to BCD converter

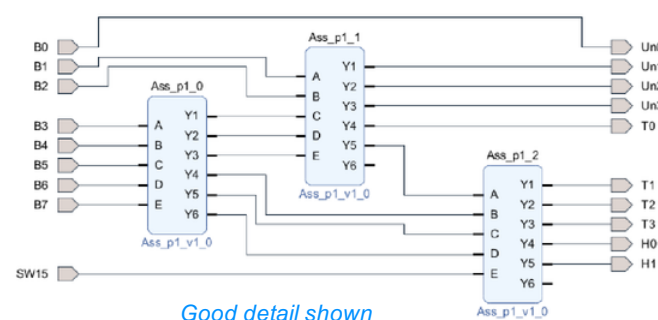


Figure 3-5

Through the use of IP block diagrams, a design source is no longer needed. However, a new test bench file is required to test this new circuit. This can be seen in **Figure 3-6**, **Figure 3-7** and **Figure 3-8**.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Binary_BCD_8bit_tb is
4  -- Port ( );
5  end Binary_BCD_8bit_tb;
6  architecture Behavioral of Binary_BCD_8bit_tb is
7  -- Component Declaration for the Unit Under Test (UUT)
8  COMPONENT Binary_BCD_8bit_wrapper
9  Port ( B0,B1,B2,B3,B4,B5,B6,B7,SW15 : in std_logic;
10 Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1 : out std_logic
11 );
12 END COMPONENT;
13
14 --Inputs
15 signal B0 : std_logic := '0';
16 signal B1 : std_logic := '1';
17 signal B2 : std_logic := '1';
18 signal B3 : std_logic := '1';
19 signal B4 : std_logic := '0';
20 signal B5 : std_logic := '1';
21 signal B6 : std_logic := '1';
22 signal B7 : std_logic := '1';
23 -- First test @ 238
24 signal SW15 : std_logic := '0';
25
26 --Outputs
27 signal Un0 : std_logic;
28 signal Un1 : std_logic;
29 signal Un2 : std_logic;
30 signal Un3 : std_logic;
31 signal T0 : std_logic;
32 signal T1 : std_logic;
33 signal T2 : std_logic;
34 signal T3 : std_logic;
35 signal H0 : std_logic;
36 signal H1 : std_logic;
37
38 --Clocks
39 -- No Clocks
40
41 begin
42 -- Instantiate the Unit Under Test (UUT)
43 uut: Binary_BCD_8bit_wrapper
44 PORT MAP (
45 --

```

Figure 3-6

```

42 : -- Instantiate the Unit Under Test (UUT)
43 : uut: Binary_BCD_8bit_wrapper
44 : PORT MAP (
45 :   B0 => B0,
46 :   B1 => B1,
47 :   B2 => B2,
48 :   B3 => B3,
49 :   B4 => B4,
50 :   B5 => B5,
51 :   B6 => B6,
52 :   B7 => B7,
53 :   SW15 => SW15,
54 :   Un0 => Un0,
55 :   Un1 => Un1,
56 :   Un2 => Un2,
57 :   Un3 => Un3,
58 :   T0 => T0,
59 :   T1 => T1,
60 :   T2 => T2,
61 :   T3 => T3,
62 :   H0 => H0,
63 :   H1 => H1
64 : );
65 : -- Clock process definitions
66 : -- No Clocks
67 : -- Stimulus process
68 : stim_proc: process
69 : begin
70 :

```

Figure 3-7

```

65 : -- Clock process definitions
66 : -- No Clocks
67 : -- Stimulus process
68 : stim_proc: process
69 : begin
70 :
71 :   wait for 1ms;
72 :   -- Second test @ 19
73 :   B0 <= '1';
74 :   B1 <= '1';
75 :   B2 <= '0';
76 :   B3 <= '0';
77 :   B4 <= '1';
78 :   B5 <= '0';
79 :   B6 <= '0';
80 :   B7 <= '0';
81 :   wait for 1ms;
82 :   -- Third test @ 162
83 :   B0 <= '0';
84 :   B1 <= '1';
85 :   B2 <= '0';
86 :   B3 <= '0';
87 :   B4 <= '0';
88 :   B5 <= '1';
89 :   B6 <= '0';
90 :   B7 <= '1';
91 :
92 :   wait;
93 : end process;
94 : end Behavioral;

```

Figure 3-8

This test bench starts the same as the one for the 6 bit BCD converter with the entity and library declaration. Following the entity is again the architecture. However, in this test bench there are many more inputs and outputs. The inputs are B0 to B7 which represent the 8 bits of binary code, and SW15 which acts as a ground for the circuit. The outputs are Un0, Un1, Un2, Un3, T0, T1, T2, T3, H0 and H1. The function of the BCD converter is to convert binary to decimal, and then display each digit of the decimal value in binary. The Un outputs display the unit digit in binary, the T outputs display the tens unit in binary, and the H output displays the hundreds unit in binary. There

are 4 unit and ten outputs due to the fact they need to go up to 9, whereas the hundreds only needs to go up to 2. This is because the maximum output of an 8 bit BCD converter is 255.

Following this the ports are defined. This test will be done differently since it is much easier check the operation of the converter by searching for specific values rather than getting a full set of values. Therefore, the inputs are defaulted to a binary value of 238 which is 11101110. Following this SW15 is defaulted to 0 as it is acting as a ground. The outputs are then set to act with STD_LOGIC.

Just like before, the port map is produced which links the inputs and outputs declared previously, to the inputs and outputs used within the component being tested. With the port map complete, the testing process is initiated. Since the test is to find 3 different values, and the first of which has already been set on the inputs, the process section only contains two sets of binary values set to the inputs. These two values are 19 (11001000) and 162 (01000101). Each set of binary values is set to change on the inputs after 1ms. Therefore a minimum of 4ms can be used to achieve each desired result in the simulation.

With the test bench completed the simulation can be done. Once again this can be found in **Section 4** along with the 6 bit BCD simulation results.

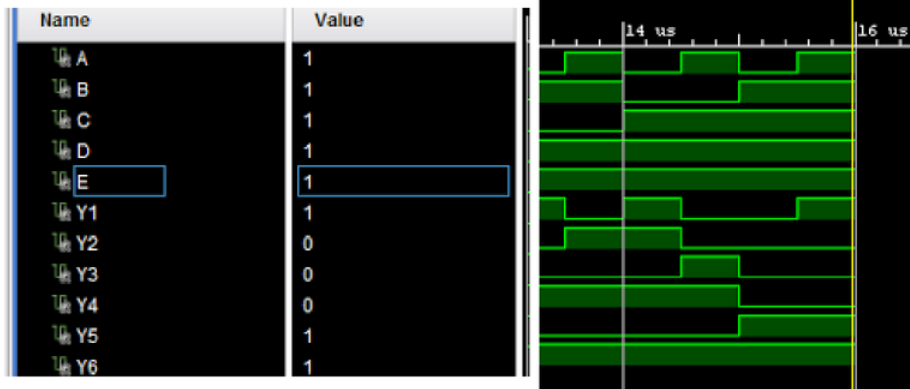
Very good explanation given which shows good understanding of the design and simulation process

4 ANALYSIS OF XILINX VIVADO SIMULATED RESULTS

This section will detail the simulated testing of both the 6 bit binary to BCD converter and the 8 bit binary to BCD converter.

4.1 6 BIT BINARY TO BCD CONVERTER SIMULATED RESULTS

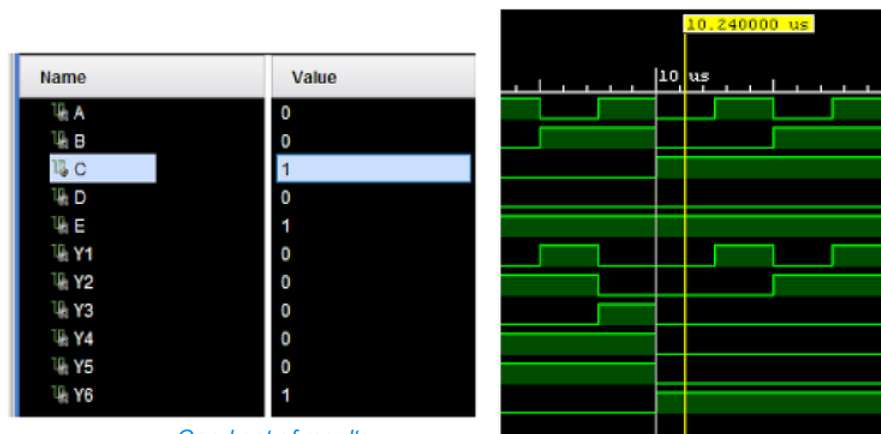
The simulated testing of the 6 bit binary to BCD converter involves looking at a complete set of results and finding 3 values within them. The three values used in these tests are 63 (111111), 40 (101000) and 19 (010011). The clock pulse acts as the (LSB) for both the binary input and the BCD output.. *Good*



Good set of results

Figure 4-1

The first test was to find 63. Since it's the highest value possible from the 6 bit BCD converter, it must happen at the end of the test which is after 16us have elapsed. As shown in **Figure 4-1**, at 16us the binary value on the inputs is 111111 which is equal to 63. The value on the outputs is 1100011 which is the BCD value for 63. The units in this BCD value are 0011 which is equal to 3 and the tens are equal to 110 which is equal to 6. Therefore the final BCD value is 63.



Good set of results

Figure 4-2

The second test is to find 40. The binary value shown in the simulation is 101000 which is equal to 40. The BCD value on the outputs is 1000000 which is again equal to 40. The units of this output are 0000 which is equal to 0, and the tens are equal to 100, which is equal to the BCD value of 4. Therefore the final BCD value is 40.

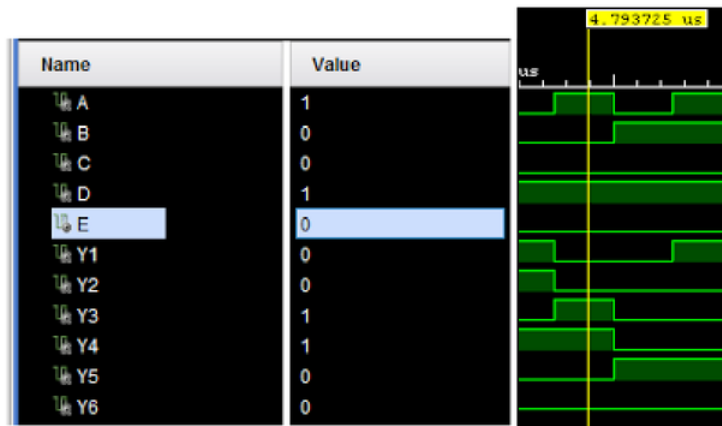


Figure 4-3

The binary value on the inputs is 010011 which is equal to 19. The value on the outputs can be seen as 0011001. The units in this binary code are 1001 which is equal to 9. The tens are 001 which is equal to 1, therefore the BCD output is 19.

With all 3 6 bit BCD converter tests being a success, the 8 bit BCD converter could be made.

Good set of test results for the 6 bit device which have been analysed well.

4.2 8 BIT BINARY TO BCD CONVERTER SIMULATED RESULTS

As previously stated, the 8 bit binary BCD converter will be tested to search for three specific values only. These three values are 238 (11101110), 19 (00010011) and 162 (10100010).

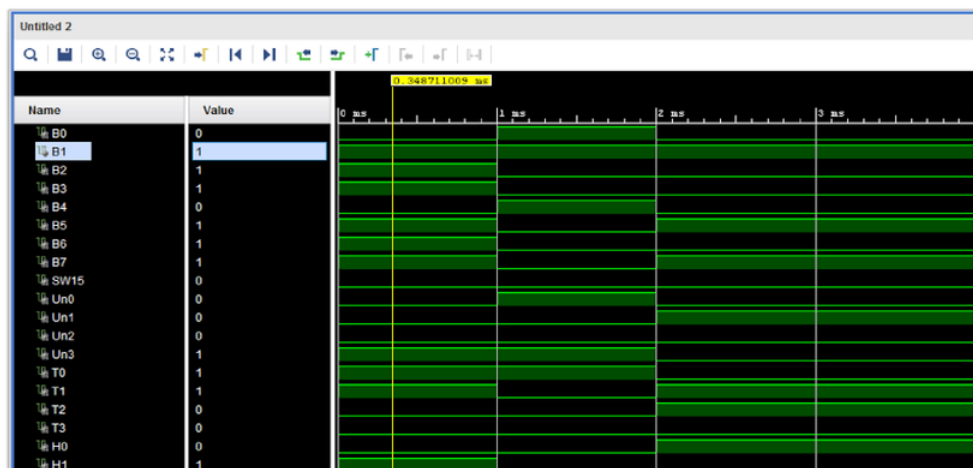


Figure 4-4

As shown above, the binary value on the input is 11101110 which is equal to 238. The binary value on the outputs is 10 for the hundreds which is equal to 2, 0011 for the tens which is equal to 3, and 1000 on the units which is equal to 8. Therefore the final BCD output for this test is 238 which is what was expected.

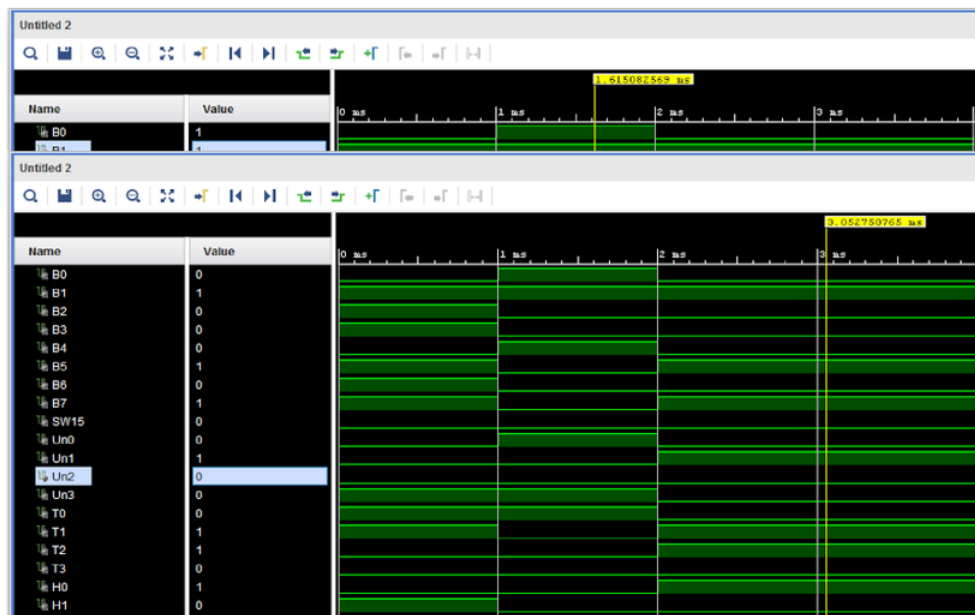
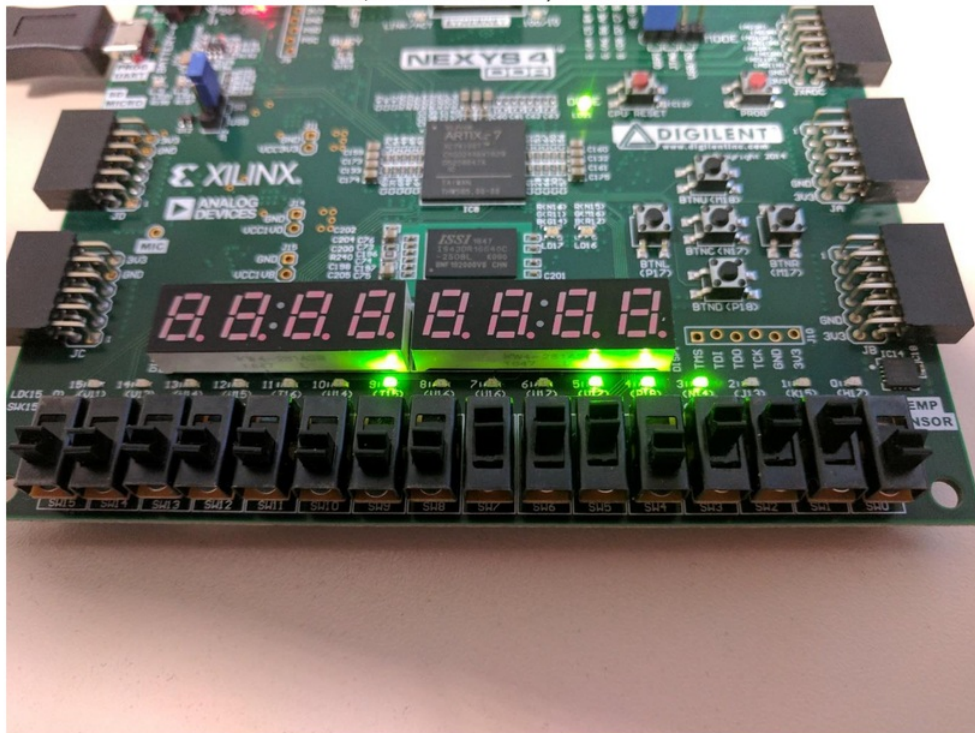


Figure 4-6

As shown above, the binary value on the input is 00010011 which is equal to 19. The binary value on the outputs is 00 for the hundreds which is equal to 0, 0001 on the tens which is equal to 1 and 1001 on the units which is equal to 9. Therefore the final BCD output for this test is 019 which was again an expected result.

*Seems to be a problem with Figure 4-5
Good analysis of the 2 (not 3!) results shown
in the report.*

The final test is shown above, which has a binary value of 10100010 on the



inputs which are equal to 162. The binary value on the outputs is 01 for the hundreds which is equal to 1, 0110 for tens which is equal to 6, and 0010 which is equal to 2. Therefore the final binary value is equal to 162 which is once again an expected result. With all of these tests proving that the 8 bit BCD converter works as intended, a practical test can be carried out on the Nexys 4 board. These practical tests can be seen in **Section 5**.

Not relevant at this point in the assignment!

5 ANALYSIS OF NEXYS4 FPGA PRACTICAL RESULTS

In order to test the 8 bit binary BCD converter on the Nexys 4 board, constraints must be added to the VHDL file. This is done by taking lines from the Nexys 4 DDR Master xdc file. The lines needed to test this VHDL file on the board are the switches and the LEDs. When importing these lines into Vivado, the input and output names must also be changed. The constraints file can be seen in

```
3 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { B0 }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
4 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { B1 }]; #IO_L3N_T0_DQS_EMCLK_14 Sch=sv[1]
5 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
6 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { B3 }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
7 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { B4 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
8 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { B5 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
9 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { B6 }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
10 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { B7 }]; #IO_L5N_T0_D07_14 Sch=sv[7]
11 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { SW15 }]; #IO_L24N_T3_34 Sch=sv[8]
12
13 ## LEDs
14
15 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Un0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
16 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { Un1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
17 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { Un2 }]; #IO_L17N_T2_A25_15 Sch=led[2]
18 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { Un3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
19 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { T0 }]; #IO_L7P_T1_D09_14 Sch=led[4]
20 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { T1 }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
21 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { T2 }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
22 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { T3 }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
23 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { H0 }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
24 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { H1 }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
```

Figure 5-1

With the constraints file complete, the VHDL file can be synthesized, implemented and a bit stream can be generated. Then finally, the VHDL file can be programmed into the Nexys 4 board for testing. The tests can be seen below for each of the 3 values used to test the 8 bit BCD converter.

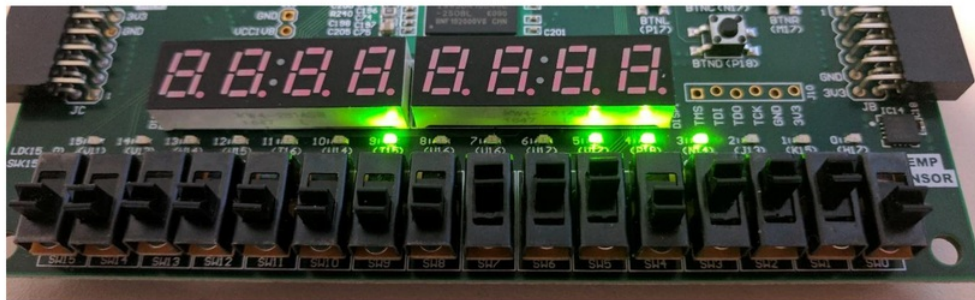


Figure 5-2

Figure 5-2 shows that the switches are set in a binary configuration of 11101110 which is equal to 238. The LEDs show 10 on the hundreds which is equal to 2, 0011 on the tens which is equal to 3, and 1000 on the units which is equal to 8. Therefore the output is 238 as expected.

Good use made of pictorial evidence

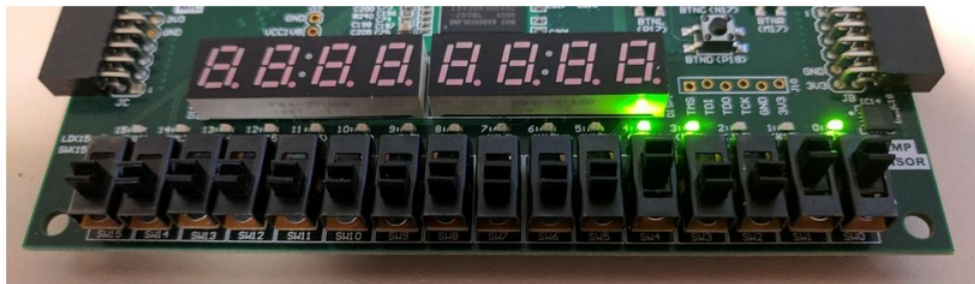


Figure 5-3

Figure 5-3 shows the switches in a configuration of 00010011 which is equal to 19. The LEDs have 00 on the hundreds, 0001 on the tens which equals 1 and 1001 on the units which equals 9. Therefore the output is an expected 19.

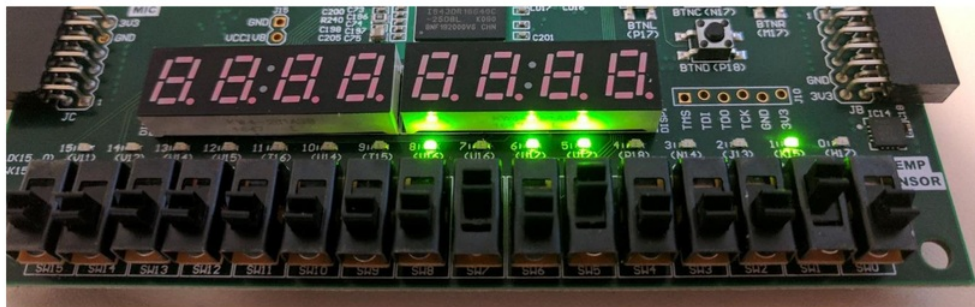


Figure 5-4

Figure 5-4 shows the switches in a configuration of 10100010 which represents 162. The LEDs show a 01 on the hundreds which equals 1, 0110 on the tens which equals 6, and 0010 on the units which equals 2. Combined they provide an output of 162 which is the expected result.

Each of these tests proved successful which means the design of both the 6 bit and 8 bit binary to BCD converters operate as intended and can be used for future designs.

3 sets of test results are correct analysed well

6 ADAPTING DESIGN TO SUIT NEW PURPOSES OR APPLICATIONS

In order to adapt the design to suit the specified purpose the block diagram from the brief must once again be looked at.

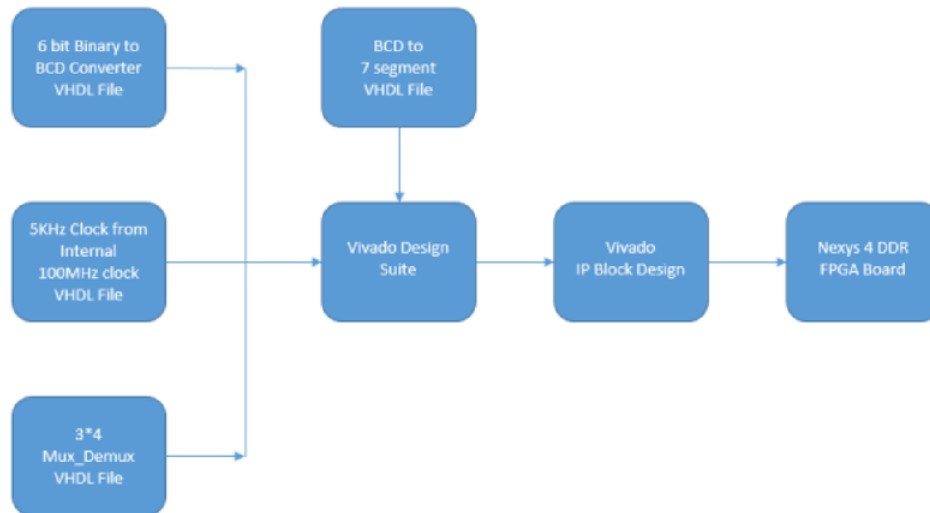


Figure 6-1

The remaining VHDL files that needs to be created in order for this final design to operate are the 5kHz Clock, BCD to 7 Segment, and 3*4 Mux_Demux. The design of these files will be detailed in this section.

6.1 5KHZ CLOCK FROM INTERNAL 100MHZ CLOCK

The 5kHz clock can be derived from the Nexys 4 100MHz internal clock by doing some calculations.

$$100\text{MHz} \rightarrow \text{PERIOD} = \frac{1}{100 \times 10^{-6}} = 10\text{ns} = 10 \times 10^{-9}$$

$$\text{TEMP1} = 10000 \times (10 \times 10^{-9}) = (10 \times 10^3) \times (10 \times 10^{-9}) = 100 \times 10^{-6} = 100\mu\text{s}$$

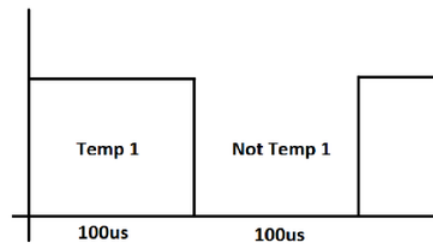


Figure 6-2

Very good design shown

$$P = 200\mu s$$

$$F = \frac{1}{200} \times 10^{-6} = 5KHz$$

With these calculations completed, the Design Source code can be written for the 5KHz clock. This can be seen below.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  entity Clock_5kHz is
6  Port ( CLK100MHZ : in STD_LOGIC;
7        Rst : in std_logic;
8        FiveKHz : out STD_LOGIC
9  );
10 end Clock_5kHz;
11 architecture Behavioral of Clock_5kHz is
12 signal cnt1 : integer := 0;
13 signal temp1 : std_logic := '0';
14 begin
15 process(CLK100MHZ,Rst)
16 begin
17 if Rst = '1' then
18 temp1 <= '0';
19 cnt1 <= 0;
20 elsif rising_edge (CLK100MHZ) then
21 if(cnt1= 10000) then
22 temp1 <= not temp1;
23 cnt1 <= 0;
24 else
25 cnt1 <= cnt1 +1 ;
26 end if;
27 end if;
28 end process;
29 FiveKHz <= temp1;
30
31 end Behavioral;

```

Figure 6-3

As always, the file starts with the library and package declarations followed by the entity. This time the entity is named Clock_5kHz, the inputs are the internal 100MHz clock and a reset (Rst), and the output is FiveKHz. The architecture begins by creating two internal signals; cnt1 which is configured as an integer and set to 0, and temp1 which is configured to std_logic and set to 0. The process is then began where if rst is equal to 1, then temp1 and cnt1 are both set to 0. Else, on the rising edge of the CLK100MHz the following code is executed. The reset is stated before the clock as it is asynchronous and should not rely on a clock pulse to reset. If cnt1 is equal to 10000, then temp1 is set to not temp1 and cnt is set back to 0. If cnt has not reached 10000 yet, then cnt is incremented by 1 each rising edge. The if statements are then ended along with the process. The output FiveKHz is then set to equal temp1 which will pulse at 5KHz, therefore creating a 5KHz clock.

Very good detail!

In order to simulate this, a test bench file must once again be created; this can be found in **Figure 6-4**.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  -- Uncomment the following library declaration if instantiating
6  entity Clock_5kHz_tb is
7  -- Port ( );
8  end Clock_5kHz_tb;
9  architecture Behavioral of Clock_5kHz_tb is
10 Component Clock_5kHz
11 Port ( CLK100MHZ : in STD_LOGIC;
12       Rst : in STD_LOGIC;
13       FiveKHz : out STD_LOGIC
14 );
15 end Component;
16 --Inputs
17 signal CLK100MHZ : STD_LOGIC := '0';
18 signal Rst : STD_LOGIC := '1';
19
20 --Outputs
21 signal FiveKHz : STD_LOGIC;
22
23 --Clock definitions
24 constant CLK100MHZ_period : time := 10ns;
25 begin
26 --Instantiate the UUT
27 UUT : Clock_5kHz
28
29 Port Map ( CLK100MHZ => CLK100MHZ,
30           Rst => Rst,
31           FiveKHz => FiveKHz

```

Figure 6-4

```

30
31 Port Map ( CLK100MHZ => CLK100MHZ,
32           Rst => Rst,
33           FiveKHz => FiveKHz
34 );
35 --Clock Process definitions
36 CLK100MHZ_process : process
37 begin
38 CLK100MHZ <= '0';
39 wait for CLK100MHZ_period/2;
40 CLK100MHZ <= '1';
41 wait for CLK100MHZ_period/2;
42 end process;
43 --Stimulus process definitions
44 stim_proc: process
45 begin
46 wait for 1ms;
47
48 Rst <= '0';
49 wait for 2ms;
50 Rst <= '1';
51
52 wait;
53 end process;
54 end Behavioral;

```

Figure 6-5

Just like the previous test bench files, this one starts with the library, package and entity declarations. The inputs and outputs are both declared in the component section. When configuring the parameters for each input, rst is set to 1 so that the clock is reset from the start of the simulation. Under clock definitions, the CLK100MHz_Period is set to 10ns which is the period of the 100MHz clock. The port map is then completed and the clock process definitions begins. CLK100MHz is set to 0, then a wait for command is issued telling the simulation to wait for half CLK100MHz_Period. After that time, CLK100MHz is then set to 1 and the same wait command is issued. Finally, the stimulus process definitions are stated which are used to alter values of the testing parameters, in this case rst is set to 1 after a wait of 2ms. The process is then ended along with behavioural.

Good understanding shown

The testing results can be found in **Section 7**.

6.2 MUX_DEMUX

The Mux_Demux circuit is used to take the binary output BCD value from the BCD converters, and relay the units, tens and hundreds at separate times to the BCD 7 segment display. The Design Source code can be seen below.

Good

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  entity Mux_Demux is
6  Port ( FiveKHz : in STD_LOGIC;
7        Rst : in STD_LOGIC;
8        AN0,AN1,AN2 : out STD_LOGIC;
9        Un0,Un1,Un2,Un3 : in STD_LOGIC;
10       T0,T1,T2,T3 : in STD_LOGIC;
11       H0,H1 : in STD_LOGIC;
12       A,B,C,D : out STD_LOGIC
13     );
14 end Mux_Demux;
15 architecture Behavioral of Mux_Demux is
16   signal cnt : std_logic_vector (1 downto 0);
17   signal Q : std_logic_vector (2 downto 0);
18   begin
19     process (FiveKHz, Rst)
20     begin
21       if Rst = '1' then
22         cnt <= "11";
23       elsif rising_edge (FiveKHz) then
24         cnt <= cnt + "01";
25       end if;
26     end process;
27
28     process (cnt(1), cnt(0))
29     begin
30       if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
31       elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
32       elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
33       else Q <= "111";
34     end if;
35   end process;
36 end Behavioral;

```

Figure 6-6

```

41 AN0 <= Q(0);
42 AN1 <= Q(1);
43 AN2 <= Q(2);
44
45 process (Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1,Q(0),Q(1),Q(2))
46 begin
47
48   if (Q(0) = '0' and Un0 = '0') then Dc='0';
49   elsif (Q(0) = '0' and Un0 = '1') then Dc='1';
50   elsif (Q(1) = '0' and T0 = '0') then Dc='0';
51   elsif (Q(1) = '0' and T0 = '1') then Dc='1';
52   elsif (Q(2) = '0' and H0 = '0') then Dc='0';
53   elsif (Q(2) = '0' and H0 = '1') then Dc='1';
54
55   end if;
56
57   if (Q(0) = '0' and Un1 = '0') then Cc='0';
58   elsif (Q(0) = '0' and Un1 = '1') then Cc='1';
59   elsif (Q(1) = '0' and T1 = '0') then Cc='0';
60   elsif (Q(1) = '0' and T1 = '1') then Cc='1';
61   elsif (Q(2) = '0' and H1 = '0') then Cc='0';
62   elsif (Q(2) = '0' and H1 = '1') then Cc='1';
63
64   end if;
65
66   if (Q(0) = '0' and Un2 = '0') then Bc='0';
67   elsif (Q(0) = '0' and Un2 = '1') then Bc='1';
68   elsif (Q(1) = '0' and T2 = '0') then Bc='0';
69   elsif (Q(1) = '0' and T2 = '1') then Bc='1';
70   elsif (Q(2) = '0') then Bc='0';
71
72   end if;
73
74   if (Q(0) = '0' and Un3 = '0') then Ac='0';
75   elsif (Q(0) = '0' and Un3 = '1') then Ac='1';
76   elsif (Q(1) = '0' and T3 = '0') then Ac='0';
77   elsif (Q(1) = '0' and T3 = '1') then Ac='1';
78   elsif (Q(2) = '0') then Ac='0';
79
80   end if;
81 end process;
82 end Behavioral;

```

Figure 6-7

Once again at the start of the VHDL file the library, packages and entity are all declared. Within the entity the inputs FiveKHz, Rst, Un0 to Un3, T0 to T3 and H0 to H1 are all declared. FiveKHz is the input for the 5KHz clock that was previously designed. The Rst is used as a reset input for the circuit. The Un inputs represent the unit inputs from the BCD converters, the T inputs represent the tens, and the H inputs represent the hundreds. The outputs declared within the entity are the AN0 to AN2 outputs, and the A, B, C, D outputs. The AN outputs are used to activate each individual seven segment display which will be later explained in the 7 seg design section. The A, B, C and D outputs represent the binary output code from the Mux_Demux which will then be sent to the 7 segment displays themselves.

The entity is then ended with the keyword *end* and the architecture begins. The architecture defines two signals, cnt and Q. The signal cnt is assigned two possible values with the line 1 downto 0. The Signal Q is assigned three possible values with the line 2 down to 0. Following this the process is began. The process takes in the FiveKHz and Rst inputs.

The first line of the process states that if Rst is equal to 1 then cnt set to "11". This is before any mention of the clock because just like in the 5KHz clock, the Rst input needs to act asynchronously so that it can be activated without relying on a rising edge of the clock pulse. The next line states that else if the clock is on a rising edge, then cnt gets 01 added to it. This process is then ended.

Another process follows which takes the signal cnt(1) and cnt(0) in. The process contains an if statements that states if cnt(1) is equal to 0 and cnt(0) is equal to 0, then Q is set to 110. The next two lines do a similar thing for increasing values of cnt(1) and cnt(0). The results output 101 and 011

as the values increase. This is then followed by a line that states that if the combination of cnt(1) and cnt(0) is anything else, then the value of Q will be equal to 111. This process is then ended.

The outputs AN0 to AN2 are then assigned values stored within Q. This allows the Mux_Demux to enable one seven segment display at a time depending on which value of Q is active. This can be further seen in the following process. [Very good detail](#)

The following process contains four sets of if and elseif statements. These sections each determine what gets sent to the 7 segment display VHDL circuit. The first set of statements begins with an if statement which states: if(Q(0)=0 and Un0=0) then D <= 0. This means that under these conditions the binary output for the D bit is 0. However the following statement states: if(Q(0)=0 and Un0=1) then D <= 1. The difference here is that Un0 is equal to 1, which means the output to the seven segment display should be 1 for the D bit. Ignoring the rest of this set for now, the next set of statements has a similar statement to begin with: if(Q(0)=0 and Un1=0) then C <= 0 which is then followed by if(Q(0)=0 and Un1=1) then C <= 1. Therefore this set of statements controls the C bit. The next set controls the B bit and the final set controls the A bit. From this it can be seen that when Q(0)=0, the units are being sent to the seven segment display through the A, B, C and D bits where D is the LSB. AN0 is also set to 0 when Q(0)=0 which means that the first seven segment display is active to display the units at the same time.

Going back to the first set. The following lines after the unit lines states: elseif (Q(1)=0 and T0=0) then D <= 0 which is then followed by elseif (Q(1)=0 and T0=1) then D <= 1. This uses the same principle as the units where Q(1) activates the ten seven segment display, and T0 determines whether the output bit is a 1 or 0. This is repeated in the three following sets of statements for C, B and A.

Finally, the last lines to discuss in the first two sets of statements act the exact same, except Q(2) controls the hundreds seven segment display and H0 and H1 determine whether the D and C bit are 1 or 0 respectively. There is only need for two 'hundred' lines of code since the maximum number that can be produced from the BCD converter is 255. So in the third and fourth set of statements, if Q(2) is set to 0, then B and A are automatically defaulted to 0 too so they'd never have an effect on the hundreds output value. [Very good detail](#)

The process is finally ended along with the architecture. In order to test the Mux_Demux, a test bench file must once again be produced. This can be seen as follows.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  entity Mux_Demux is
6  Port ( FiveKHz : in STD_LOGIC;
7        Rst : in STD_LOGIC;
8        AN0,AN1,AN2 : out STD_LOGIC;
9        Un0,Un1,Un2,Un3 : in STD_LOGIC;
10       T0,T1,T2,T3 : in STD_LOGIC;
11       H0,H1 : in STD_LOGIC;
12       A,B,C,D : out STD_LOGIC
13     );
14 end Mux_Demux;
15 architecture Behavioral of Mux_Demux is
16 signal cnt : std_logic_vector (1 downto 0);
17 signal Q : std_logic_vector (2 downto 0);
18 begin
19   process (FiveKHz, Rst)
20   begin
21     if Rst = '1' then
22       cnt <= "11";
23     elsif rising_edge (FiveKHz) then
24       cnt <= cnt + "01";
25     end if;
26   end process;
27
28   process (cnt(1), cnt(0))
29   begin
30     if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
31     elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
32     elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
33   end process;
34 end Behavioral;
35

```

Figure 6-8

```

41  AN0 <= Q(0);
42  AN1 <= Q(1);
43  AN2 <= Q(2);
44
45  process (Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1,Q(0),Q(1),Q(2))
46  begin
47
48    if (Q(0) = '0' and Un0 = '0') then Dc<='0';
49    elsif (Q(0) = '0' and Un0 = '1') then Dc<='1';
50    elsif (Q(1) = '0' and T0 = '0') then Dc<='0';
51    elsif (Q(1) = '0' and T0 = '1') then Dc<='1';
52    elsif (Q(2) = '0' and H0 = '0') then Dc<='0';
53    elsif (Q(2) = '0' and H0 = '1') then Dc<='1';
54  end if;
55
56    if (Q(0) = '0' and Un1 = '0') then Cc<='0';
57    elsif (Q(0) = '0' and Un1 = '1') then Cc<='1';
58    elsif (Q(1) = '0' and T1 = '0') then Cc<='0';
59    elsif (Q(1) = '0' and T1 = '1') then Cc<='1';
60    elsif (Q(2) = '0' and H1 = '0') then Cc<='0';
61    elsif (Q(2) = '0' and H1 = '1') then Cc<='1';
62  end if;
63
64    if (Q(0) = '0' and Un2 = '0') then Bc<='0';
65    elsif (Q(0) = '0' and Un2 = '1') then Bc<='1';
66    elsif (Q(1) = '0' and T2 = '0') then Bc<='0';
67    elsif (Q(1) = '0' and T2 = '1') then Bc<='1';
68    elsif (Q(2) = '0') then Bc<='0';
69  end if;
70
71    if (Q(0) = '0' and Un3 = '0') then Ac<='0';
72    elsif (Q(0) = '0' and Un3 = '1') then Ac<='1';
73    elsif (Q(1) = '0' and T3 = '0') then Ac<='0';
74    elsif (Q(1) = '0' and T3 = '1') then Ac<='1';
75    elsif (Q(2) = '0') then Ac<='0';
76  end if;
77
78  end process;
79 end Behavioral;
80

```

Figure 6-9

From the test bench file shown above, it can be seen that all the inputs and outputs are declared as normal. The testing process is identical to that of the design source as this will provide the results needed to test the VHDL file.

6.3 SEVEN SEGMENT DISPLAY

The final file to be created in order to fulfil the specification is the BCD to Seven Segment Display VHDL file. This Design Source can be seen in **Figure 6-10**.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity BCD_7seg is
4      Port ( A,B,C,D : in std_logic;
5            sa,sb,sc,sd,se,sf,sg : out std_logic
6            );
7
8  end BCD_7seg;
9  architecture Behavioral of BCD_7seg is
10     signal Q: std_logic_vector (3 downto 0);
11     signal seg : std_logic_vector (6 downto 0);
12     begin
13         Q(0) <= D;
14         Q(1) <= C;
15         Q(2) <= B;
16         Q(3) <= A;
17     process (Q)
18     begin
19         case Q is
20             WHEN "0000" => -- 0
21                 seg <= ("0000001");
22             WHEN "0001" => -- 1
23                 seg <= ("1001111");
24             WHEN "0010" => -- 2
25                 seg <= ("0010010");
26             WHEN "0011" => -- 3
27                 seg <= ("0000110");
28             WHEN "0100" => -- 4
29                 seg <= ("1001100");
30             WHEN "0101" => -- 5
31                 seg <= ("0100100");
32             WHEN "0110" => -- 6
33                 seg <= ("0100000");
34             WHEN "0111" => -- 7
35                 seg <= ("0001111");
36             WHEN "1000" => -- 8
37                 seg <= ("0000000");
38             WHEN "1001" => -- 9
39                 seg <= ("0001100");
40             WHEN Others => -- 'blank'
41                 seg <= ("1111111");
42         END CASE;
43     END PROCESS;
44     sa <= seg(6);
45     sb <= seg(5);
46     sc <= seg(4);
47     sd <= seg(3);
48     se <= seg(2);
49     sf <= seg(1);
50     sg <= seg(0);
51 end Behavioral;

```

Where does this
information come
from?

Figure 6-10

In the entity of this file the inputs A, B, C and D are declared along with the outputs sa, sb, sc, sd, se, sf, sg. The inputs are the outputs from the Mux_Demux which represent a binary value for units, tens and hundreds. The outputs each represent a segment on a seven segment display. **Figure 6-11** shows a 7 segment display diagram labelled with each segment.

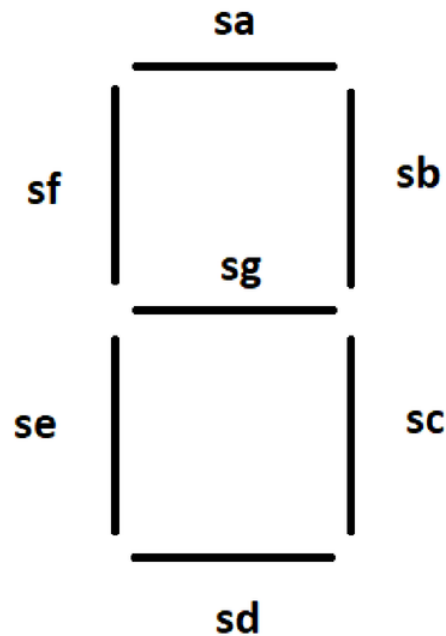


Figure 6-11

In the architecture of the VHDL file, the internal signals Q and seg are declared. Q is given 4 potential values, and seg is given 7. Q is used to store the values of A, B, C and D, and seg is used to store the binary value which will display a number on the 7 segment displays. For example to display the number 1, sb and sc must be active. The way the 7 segment displays are wired on the Nexys board means a 0 turns the segment on and a 1 turns the segment off. Therefore the binary code to make a number 1 would be 1001111 where sa is the MSB and sg is the LSB.

Since Q is assigned the binary values held in A, B, C and D, a case statement is used to determine which number should be displayed and hence sends the correct binary code for the display to the internal seg signal. For example, when Q is equal to 0101 (5) the binary 7 segment code for 5 is stored in seg.

Finally, each bit of the binary code stored in seg is assigned to the outputs as previously stated where sa is the MSB and sg is the LSB. Thus, the architecture is ended. The next step is to create a test bench file in order to test the 7 segment display. This can be seen below.

Good understanding shown

```

1 : LIBRARY ieee;
2 : USE ieee.std_logic_1164.ALL;
3 : -- Uncomment the following library declaration if using
4 : -- arithmetic functions with Signed or Unsigned values
5 : --USE ieee.numeric_std.ALL;
6 : ENTITY BCD_7seg_tb IS
7 : END BCD_7seg_tb;
8 : ARCHITECTURE behavior OF BCD_7seg_tb IS
9 :     -- Component Declaration for the Unit Under Test (UUT)
10 : COMPONENT BCD_7seg
11 :     PORT(
12 :         A : IN std_logic;
13 :         B : IN std_logic;
14 :         C : IN std_logic;
15 :         D : IN std_logic;
16 :         aa : OUT std_logic;
17 :         ab : OUT std_logic;
18 :         ac : OUT std_logic;
19 :         ad : OUT std_logic;
20 :         ae : OUT std_logic;
21 :         af : OUT std_logic;
22 :         ag : OUT std_logic
23 :     );
24 : END COMPONENT;
25 :
26 : --Inputs
27 : signal A : std_logic := '0';
28 : signal B : std_logic := '0';
29 : signal C : std_logic := '0';
30 : signal D : std_logic := '0';
31 : --Outputs
32 : signal aa : std_logic;
33 : signal ab : std_logic;
34 : signal ac : std_logic;
35 : signal ad : std_logic;
36 : signal ae : std_logic;
37 : signal af : std_logic;
38 : signal ag : std_logic;
39 : -- No clocks detected in port list. Replace <clock> below with
40 : -- appropriate port name
41 : constant A_period : time := 80 ns;
42 : constant B_period : time := 40 ns;
43 : constant C_period : time := 20 ns;
44 : constant D_period : time := 10 ns;
45 : BEGIN

```

```

44 : constant D_period : time := 10 ns;
45 BEGIN
46 -- Instantiate the Unit Under Test (UUT)
47 uut: BCD_7seg PORT MAP (
48   A => A,
49   B => B,
50   C => C,
51   D => D,
52   sa => sa,
53   sb => sb,
54   sc => sc,
55   sd => sd,
56   se => se,
57   sf => sf,
58   sg => sg
59 );
60 -- Clock process definitions
61 A_process :process
62 begin
63   A <= '0';
64   wait for A_period/2;
65   A <= '1';
66   wait for A_period/2;
67 end process;
68 B_process :process
69 begin
70   B <= '0';
71   wait for B_period/2;
72   B <= '1';
73   wait for B_period/2;
74 end process;
75 C_process :process
76 begin
77   C <= '0';
78   wait for C_period/2;
79   C <= '1';
80   wait for C_period/2;
81 end process;
82 D_process :process
83 begin
84   D <= '0';
85   wait for D_period/2;
86   D <= '1';
87   wait for D_period/2;
88 end process;
89 end behavior;
90

```

Figure 6-12

The entity of this test bench simply contains the library, package and test bench name. The architecture of the file contains the component definition for the BCD_7seg. The inputs and outputs are all declared within the component definition. The inputs are all defaulted to 0 followed by the periods of each bit being assigned a time period. A_period being the MSB has a period of 80ns in, B_Period has a period of 40 ns, C_ period has 20ns and D_period has 10ns. These time periods all have a binary relationship which allows every possible combination of number to be displayed in the simulation. The Unit Under Test is then instantiated, and a port map is created. The clock process definition follows this which defines how the simulation will be carried out.

This simulation works similar to that of the 6 bit binary to BCD converter where each bit is assigned 0 at the start, then after half their assigned period, they changed to a 1, and this loops. Due

to the binary relationship between the time periods after a full 80ns, every result will be shown in the simulation. The simulation of this file can be found in [Section 7](#). [Good detail](#)

In addition to this, for the final build constraints must be created so that the program can properly interact with the inputs and outputs on the board. This is done using the Nexys DDR 4 master xdc file. The constraints can be seen in

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { D }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { C }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sv[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { A }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]

set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { SW0 }]; #IO_L24N_T3_34 Sch=sv[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { SW1 }]; #IO_25_34 Sch=sv[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW2 }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sv[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW3 }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW4 }]; #IO_L24P_T3_35 Sch=sv[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW5 }]; #IO_L20P_T3_A06_D24_14 Sch=sv[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW6 }]; #IO_L19P_T3_A08_D25_VREF_14 Sch=sv[14]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW7 }]; #IO_L21P_T3_DQS_14 Sch=sv[15]

##7 segment display

set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { sa }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { sb }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { sc }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { sd }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { se }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { sf }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { sg }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

[Good](#)

The inputs and outputs used in this program are the switches, the 7 segment display enablers, the 7 segment display segments and LEDs. Each of the inputs and outputs used had to have their names changed within this file in order to allow the program to communicate with the Nexys board.

7 ANALYSIS OF XILINX VIVADO SIMULATED RESULTS

This section will detail the simulated testing of the 5KHz clock, Mux_Demux and the BCD_7seg VHDL files.

7.1 5KHz CLOCK SIMULATION TEST

Using the test bench designed in the previous section, the 5KHz clock can be tested. The results of this test can be seen below. An expected result would be to have a period of 200us between pulses.

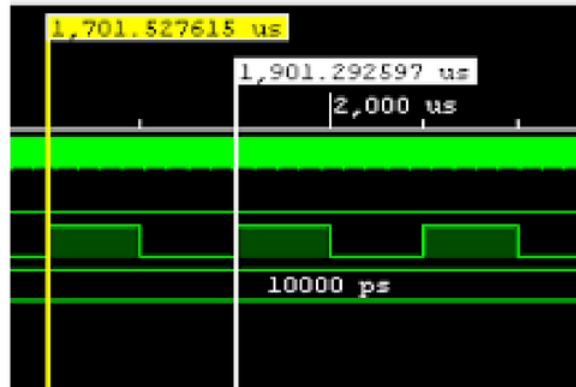


Figure 7-1

As **Figure 7-1** shows the time taken for one cycle is 200us which is the expected result.

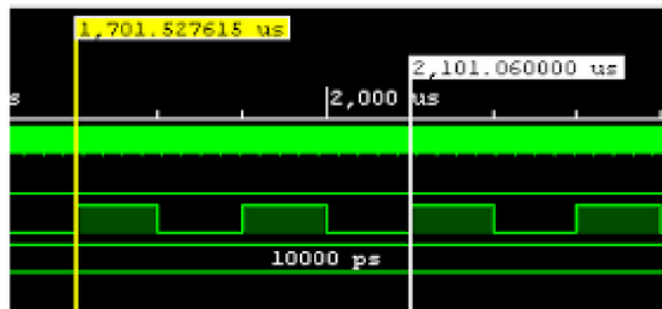


Figure 7-2

For further proof, **Figure 7-2** shows the time taken for two full cycles is 400us. This proves that the frequency provided by this clock VHDL file is at 5KHz.

Good understanding shown

7.2 MUX_DEMUX SIMULATION TEST

In order to test the Mux_Demux VHDL circuit, a number is selected and input to the test bench file. The simulation should show staggered values which indicates the binary values for units, tens and hundreds being sent to their respective 7 segment display in a cycle. The number selected is 117.

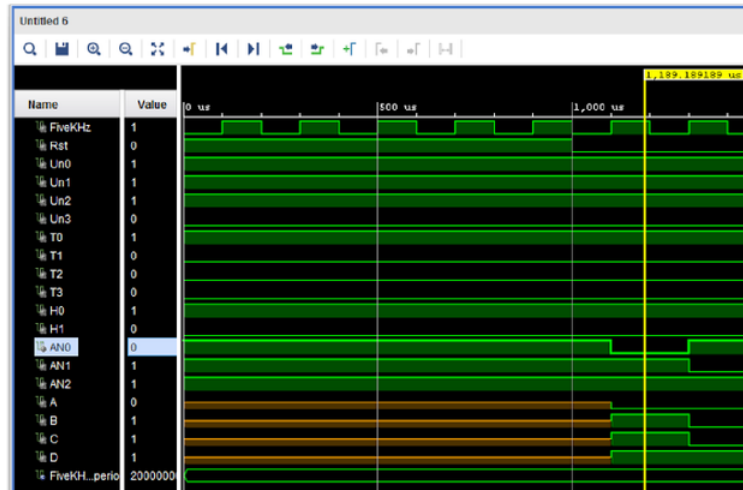


Figure 7-3

Figure 7-3 shows that AN0 is set to 0 and AN1 and AN2 are set to 1. This means the units 7 segment display is active. Below that it can be seen that the binary value sent by A, B, C and D is 0111 which is equal to 7. This is an expected result as 7 is the unit in 117. Another thing to note is that the change in values for AN0 and A, B, C and D occurs on the rising edge of the clock which was stated in the test bench file.

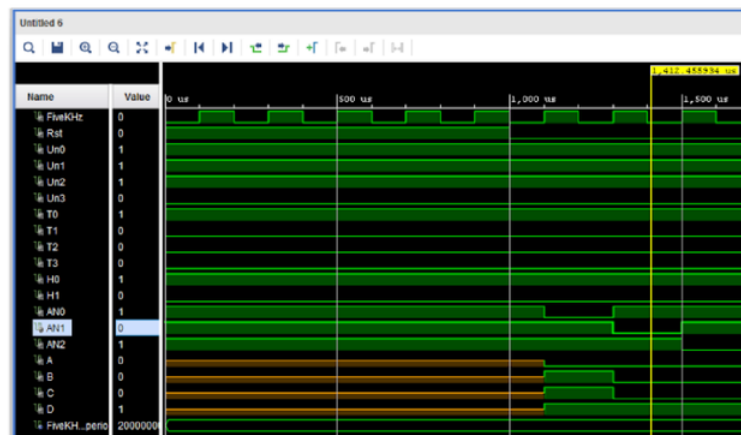


Figure 7-4

Figure 7-2 shows that AN0 has been set back to 1, and AN2 has now become 0. AN3 remains at 1. This means that the tens 7 segment display is active. The binary code sent by A, B, C and D is 0001 which is equal to 1. Once again this is expected as 1 is the tens digit in 117.

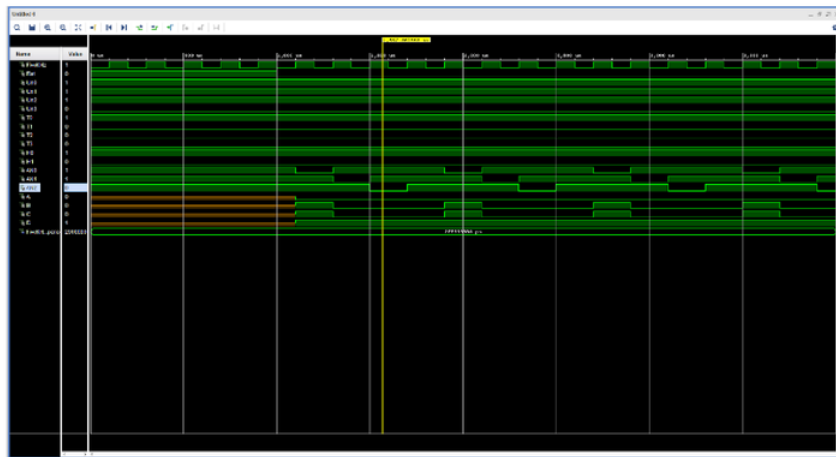


Figure 7-5

Figure 7-5 shows that A0 remains at 1, A2 has returned to 1 and A3 is now set to 0. This means the hundreds 7 segment display is active. Once again the binary code from A, B, C and D is 0001 which represents 1. This is the final expected result as 1 is the hundreds digit in 117.

Following this cycle is a single cycle where A0, A1 and A2 are all set to 1. This is again expected as the test bench said that if cnt(1) and cnt(0) are anything other than 0 0, 0 1 or 1 0 respectively then Q should be set to 111.

Good analysis of Mux/Demux

7.3 BCD_7SEG SIMULATION TEST

The final simulation test to be carried out is the BCD_7seg. In order to test this, 3 numbers that range from 0-9 are selected which should then be found in the simulation. The three numbers selected are 0, 6 and 9.

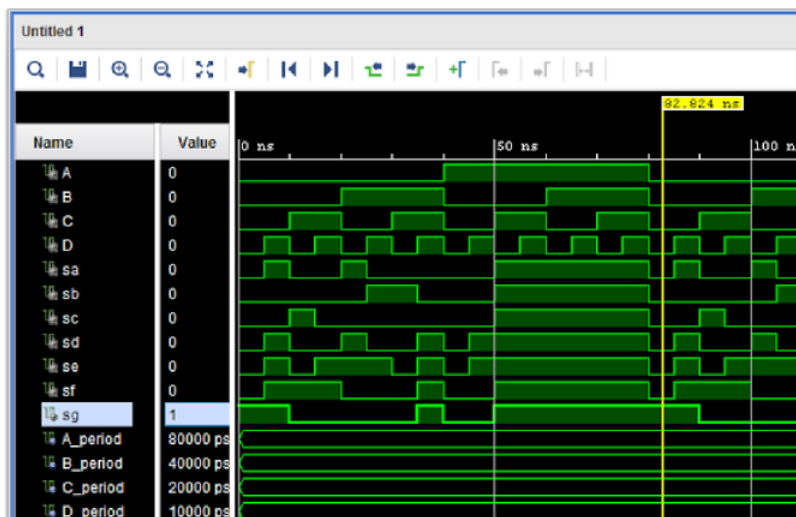


Figure 7-6

Figure 7-6 shows that at the cursor, the only segment which is active is sg. This means that the number created by the remaining segments is a 0 which was expected.

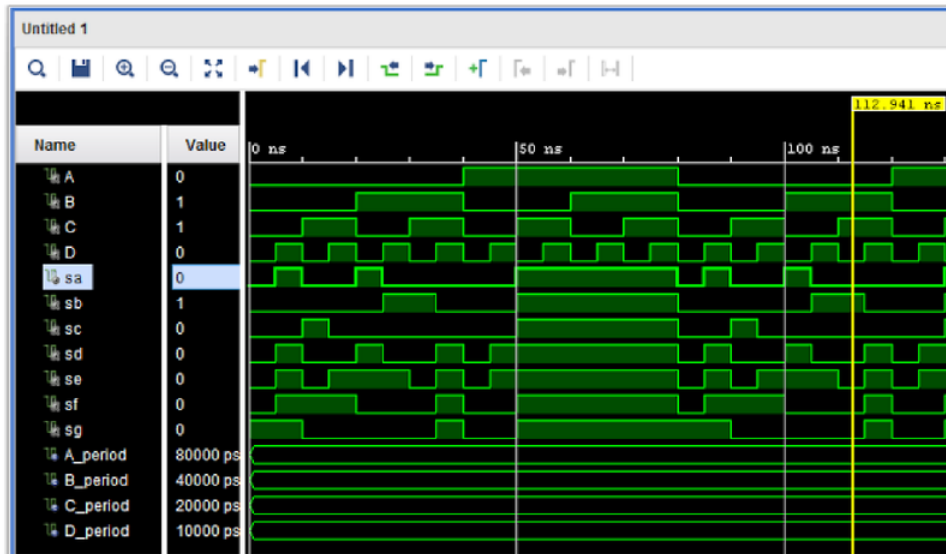


Figure 7-7

Figure 7-7 shows the cursor at a point where the binary code is 0100000. This means that sb is deactivated. When sb is deactivated and the rest of the segments are active, the number 6 is displayed on the 7 segment display. Once again this is an expected result.

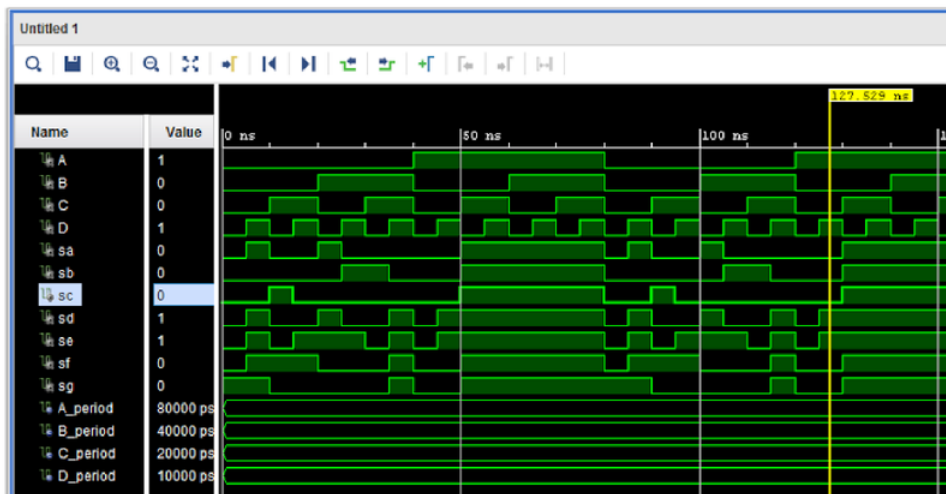


Figure 7-8

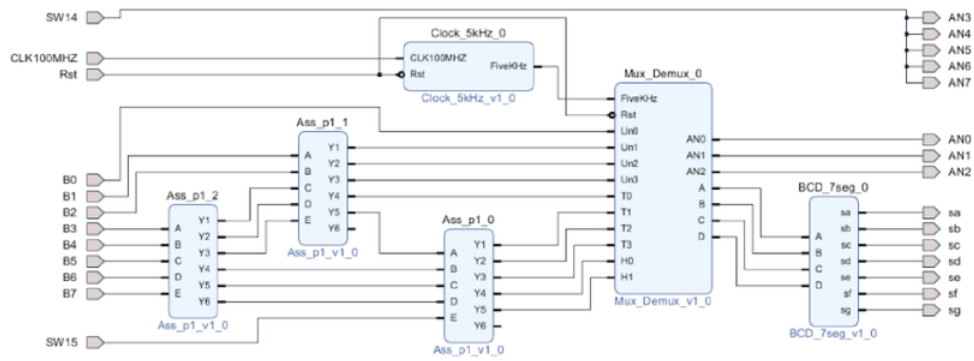
Figure 7-7 shows the cursor at a location where the binary value of the segments is 0001100. This means that segments sd and se are deactivated. This leaves the remaining segments displaying a number 9, which was the final number part of the test.

Very good understanding shown of test results

With each of the components tested and proved working, the final design can be created using IP block diagrams of each other VHDL files in a new design in Vivado.

8 ANALYSIS OF NEXYS 4 DDR FPGA PRACTICAL RESULTS

This section will detail the testing of the final design which is created using a combination of the 6 bit Binary to BCD converter, 5KHz clock, Mux_Demux and BCD_7seg. All of these VHDL files were created into IP block diagrams which were then stored within the IP repository. The final design can be seen in



Good detail on block diagram

Figure 8-1

With these components in this configuration, a Binary to BCD converter which displays the desired number on 7 segment displays is created. This design can be tested in the Nexys 4 board. An expected result would be to have a BCD value input on the switches, and the same value displayed on the 7 segment displays.

Before programming to the board, the constraints must once again be created. These can be seen

```

3 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
4 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { CLK100MHZ }];
5
6 ## Switches
7
8 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { B0 }]; #IO_L24N_T3_R50_15 Sch=sw[0]
9 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { B1 }]; #IO_L3N_T0_D05_DMCLK_14 Sch=sw[1]
10 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L6N_T0_D05_VREF_14 Sch=sw[2]
11 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { B3 }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
12 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { B4 }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
13 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { B5 }]; #IO_L7N_T1_D10_14 Sch=sw[5]
14 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { B6 }]; #IO_L7N_T2_A13_D23_14 Sch=sw[6]
15 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { B7 }]; #IO_L6N_T0_D07_14 Sch=sw[7]
16 #set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { B0 }]; #IO_L24N_T3_34 Sch=sw[8]
17 #set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { B1 }]; #IO_25_34 Sch=sw[9]
18 #set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_D05_R0WR_B_14 Sch=sw[10]
19 #set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D15_14 Sch=sw[11]
20 #set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
21 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { Rat }]; #IO_L20P_T3_A05_D24_14 Sch=sw[13]
22 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW14 }]; #IO_L10N_T3_A05_D25_VREF_14 Sch=sw[14]
23 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW15 }]; #IO_L21P_T3_D05_14 Sch=sw[15]
24
25 ## 7 segment display
26
27 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { a }]; #IO_L24N_T3_A00_D16_14 Sch=swa
28 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { ab }]; #IO_26_14 Sch=swb
29 set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { ac }]; #IO_26_15 Sch=swc
30 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { ad }]; #IO_L17P_T2_A26_15 Sch=swd
31 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { ae }]; #IO_L13P_T2_MRCC_14 Sch=swa
32 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { af }]; #IO_L19P_T3_A10_D26_14 Sch=swf
33 set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { ag }]; #IO_L4P_T0_D04_14 Sch=swg
34 #set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L15N_T3_A21_VREF_15 Sch=swdp
35
36 ## Display Controls
37
38 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
39 set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
40 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
41 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T2_A27_15 Sch=an[3]
42 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L24N_T1_D15_14 Sch=an[4]
43 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
44 set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L23P_T3_35 Sch=an[6]
45 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L23N_T2_A02_D18_14 Sch=an[7]
46

```

Figure 8-2 Good

The constraints imported from the master xdc file were the switches, 7 segment displays, and the display controls. AN0 to AN7 control whether the 7 segment display is active. This is used to deactivate the displays that are not meant to be used.

Switch 14 is used to turn off the displays that are not needed so the only three that should be active are the right most three. Switch 15 can also be used to turn off the remaining 3 displays. The tests will involve displaying the same three numbers used to test the 8 bit Binary to BCD converter, 238, 19 and 162. *Very good understanding shown on the use of SW14 & 15*

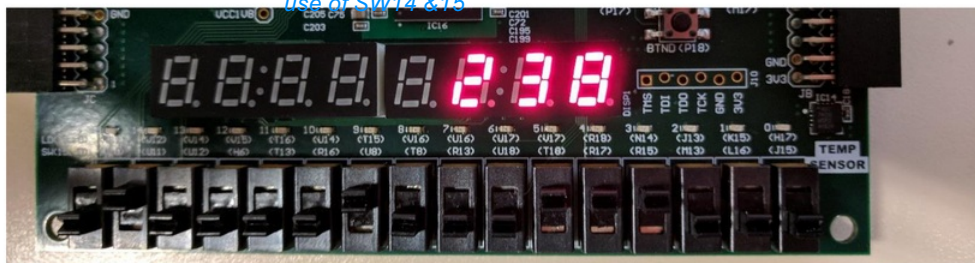


Figure 8-3

This first test shows the switches in a BCD configuration of 1000111000. This is the BCD value for 238 as the hundreds are 10 which is equal to 2, the tens are 0011 which is equal to 3 and the units are 1000 which is equal to 8.

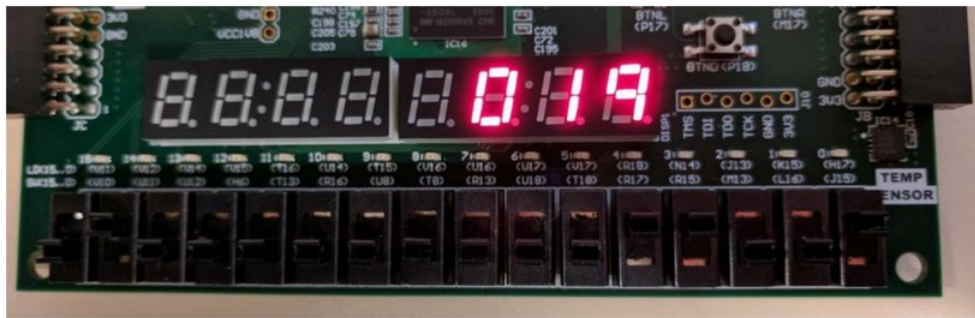


Figure 8-4

This test has a BCD configuration of 0000011001 on the switches. This is the BCD value for 19 as it has 00 on the hundreds, 0001 on the tens which is equal to 1, and 1001 on the units which is equal to 9.

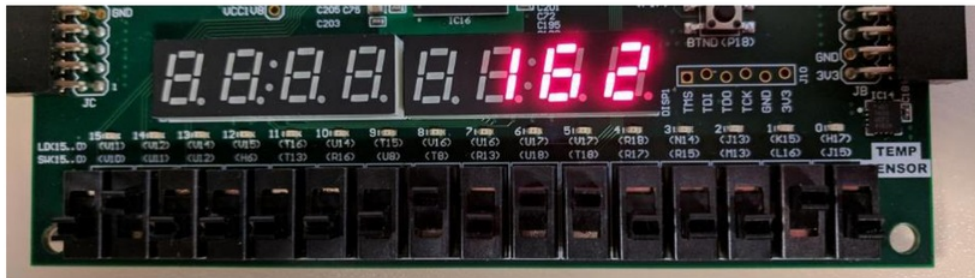


Figure 8-5

This test has a BCD configuration of 0101100010 on the switches. This is the BCD value for 162 as it has 01 on the hundreds which equals 1, 0110 on the tens which is equal to 6, and 0010 on the units which is equal to 2.

In all of these tests, the only three displays that have been on are the rightmost ones. This is because SW14 has been set to 1 in each test. When SW15 is set to 1, the current three displays also turn off. When SW14 is set to 0, the five left 7 segment displays become active. This can be seen in **Figure 8-6** and **Figure 8-7**.

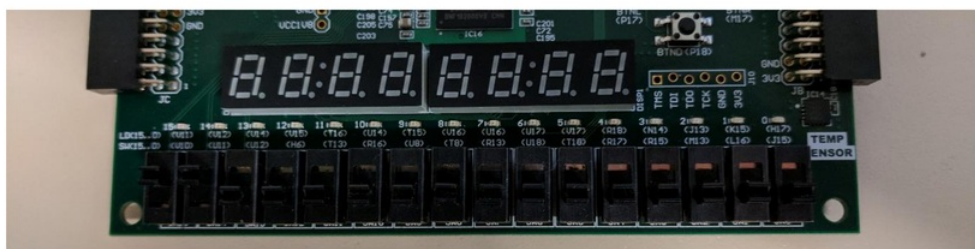


Figure 8-6

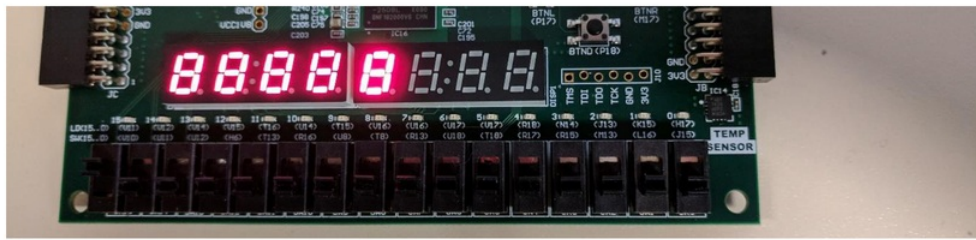


Figure 8-7

Excellent use made of pictorial evidence in analysing test results obtained

9 CONCLUSION

In conclusion, VHDL is a very powerful language that can easily create components which can later be reused as many times as needed through the use of Vivado's IP block diagram function. However, VHDL can also be much more complex than the code that was used in this assignment.

Throughout the assignment the design processes went perfectly as every simulation and practical test passed with no issue. Each of the results obtained from Task 3 matched up with the results obtained from Task 7. This proves that throughout the design process, and through the use of IP block diagrams, the function of the components still remains.

The use of the datasheet proved useful as the configuration of the 3*6 bit binary to BCD converters was provided on the sheet. Therefore time was saved deriving that configuration using Boolean logic.

The fact that each of the VHDL files could be simulated before being programmed to the board shows how versatile and useful Vivado is when it comes to the design process of these systems. Without the simulation function, it would be hard to ensure whether a design worked without programming it to a board which is extremely time consuming and tedious.

*Good general conclusion with some reflection
noted*

10 REFERENCES

3. Basic Structure of a VHDL file – Euro Electronica. 2018. 3. *Basic Structure of a VHDL file – Euro Electronica*. [ONLINE] Available at: <http://www.euroelectronica.ro/3-basic-structure-of-a-vhdl-file/>. [Accessed 01 January 2018].
- Programmable Logic/VHDL Module Structure - Wikibooks, open books for an open world. 2018. *Programmable Logic/VHDL Module Structure - Wikibooks, open books for an open world*. [ONLINE] Available at: https://en.wikibooks.org/wiki/Programmable_Logic/VHDL_Module_Structure. [Accessed 01 January 2018].
- Perry, D. (2002). *VHDL*. New York: McGraw-Hill.
- www.tutorialspoint.com. (2018). Codes Conversion. [online] Available at: https://www.tutorialspoint.com/computer_logical_organization/codes_conversion.htm [Accessed 04 Jan. 2018].
- Eng.utah.edu. (2018). Binary to BCD. [online] Available at: <http://www.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html> [Accessed 04 Jan. 2018].
- Baker, G. (2018). The std_logic type. [online] Cs.sfu.ca. Available at: https://www.cs.sfu.ca/~ggbaker/reference/std_logic/1164/std_logic.html [Accessed 05 Jan. 2018].
- DM74184/DM74185A BCD-to-Binary and Binary-to-BCD Converters. (2018). 1st ed. [ebook] Hong Kong: National Semiconductor, pp.2-10. Available at: <http://www.utm.edu/staff/leeb/DM74185.pdf> [Accessed 10 Jan. 2018].

Good, varied list of citations noted

11 APPENDICES

Bin	Binary Value						BCD Tens			BCD Units			
	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	T2	T1	T0	U3	U2	U1	U0
	E	D	C	B	A		Y6	Y5	Y4	Y3	Y2	Y1	
0/1	0	0	0	0	0	0/1	0	0	0	0	0	0	0/1
2/3	0	0	0	0	1	0/1	0	0	0	0	0	1	0/1
4/5	0	0	0	1	0	0/1	0	0	0	0	1	0	0/1
6/7	0	0	0	1	1	0/1	0	0	0	0	1	1	0/1
8/9	0	0	1	0	0	0/1	0	0	0	1	0	0	0/1
10/11	0	0	1	0	1	0/1	0	0	1	0	0	0	0/1
12/13	0	0	1	1	0	0/1	0	0	1	0	0	1	0/1
14/15	0	0	1	1	1	0/1	0	0	1	0	1	0	0/1
16/17	0	1	0	0	0	0/1	0	0	1	0	1	1	0/1
18/19	0	1	0	0	1	0/1	0	0	1	1	0	0	0/1
20/21	0	1	0	1	0	0/1	0	1	0	0	0	0	0/1
22/23	0	1	0	1	1	0/1	0	1	0	0	0	1	0/1
24/25	0	1	1	0	0	0/1	0	1	0	0	1	0	0/1
26/27	0	1	1	0	1	0/1	0	1	0	0	1	1	0/1
28/29	0	1	1	1	0	0/1	0	1	0	1	0	0	0/1
30/31	0	1	1	1	1	0/1	0	1	1	0	0	0	0/1
32/33	1	0	0	0	0	0/1	0	1	1	0	0	1	0/1
34/35	1	0	0	0	1	0/1	0	1	1	0	1	0	0/1
36/37	1	0	0	1	0	0/1	0	1	1	0	1	1	0/1
38/39	1	0	0	1	1	0/1	0	1	1	1	0	0	0/1
40/41	1	0	1	0	0	0/1	1	0	0	0	0	0	0/1
42/43	1	0	1	0	1	0/1	1	0	0	0	0	1	0/1
44/45	1	0	1	1	0	0/1	1	0	0	0	1	0	0/1
46/47	1	0	1	1	1	0/1	1	0	0	0	1	1	0/1
48/49	1	1	0	0	0	0/1	1	0	0	1	0	0	0/1
50/51	1	1	0	0	1	0/1	1	0	1	0	0	0	0/1
52/53	1	1	0	1	0	0/1	1	0	1	0	0	1	0/1
54/55	1	1	0	1	1	0/1	1	0	1	0	1	0	0/1
56/57	1	1	1	0	0	0/1	1	0	1	0	1	1	0/1
58/59	1	1	1	0	1	0/1	1	0	1	1	0	0	0/1
60/61	1	1	1	1	0	0/1	1	1	0	0	0	0	0/1
62/63	1	1	1	1	1	0/1	1	1	0	0	0	1	0/1

Binary to BCD Converter Truth Table

Sam Harries Assignment 1 - Final Version

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

68/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

Text Comment. Good detail...

PAGE 7

Text Comment. Good detail

PAGE 8

PAGE 9

PAGE 10

PAGE 11

Text Comment. Good research but not referenced?

Text Comment. Excellent!

PAGE 12

Text Comment. Very good detail

PAGE 13

Text Comment. Where is the diagram? More detail required

Text Comment. Where does this information come from?

Text Comment. Good detail shown

Text Comment. Very good explanation given which shows good understanding of the design and simulation process

Text Comment. Good

Text Comment. Good set of results

Text Comment. Good set of results

Text Comment. Good set of test results for the 6 bit device which have been analysed well

Text Comment. Seems to be a problem with Figure 4-5

Text Comment. Good analysis of the 2 (not 3!) results shown in the report.

Text Comment. Not relevant at this point in the assignment!

Text Comment. Good use made of pictorial evidence

Text Comment. 3 sets of test results are correct analysed well

Text Comment. Very good design shown

Text Comment. Very good detail!

Text Comment. Good understanding shown

Text Comment. Good

Text Comment. Very good detail

Text Comment. Very good detail

Text Comment. Where does this information come from?

Text Comment. Good understanding shown

Text Comment. Good detail

Text Comment. Good

Text Comment. Good understanding shown

PAGE 41

PAGE 42

Text Comment. Good analysis of Mux/Demux

PAGE 43

Text Comment. Very good understanding shown of test results

PAGE 44

PAGE 45

Text Comment. Good detail on block diagram

PAGE 46

Text Comment. Good

Text Comment. Very good understanding shown on the use of SW14 &15

PAGE 47

PAGE 48

Text Comment. Excellent use made of pictorial evidence in analysing test results obtained

PAGE 49

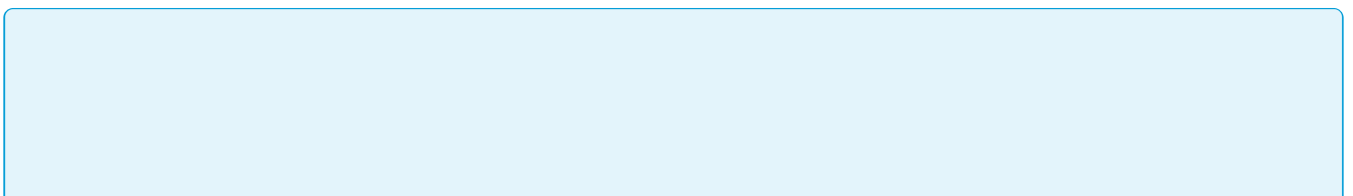
Text Comment. Good general conclusion with some reflection noted

PAGE 50

Text Comment. Good, varied list of citations noted

PAGE 51

GRADING FORM: CSG-RUBIC



RESEARCH-INFORMED LITERATURE



Structure of a typical VHDL file: Good research carried out but not referenced well. Non-standard referencing used. (2/5) Binary to BCD conversion: Excellent understanding shown. (4/5) IP Block Design in Vivado: More detail required. (1/5) Total (7/15)

7

KNOWLEDGE AND UNDERSTANDING



Design of 6/8 bit Binary to BCD Converter: 6 bit design could have been explained better but testbench explained well. (3/5). Very good explanation given for the 8 bit converter which shows good understanding of the design and simulation process. (4/5) Re-design to suit new applications: Very good design shown for 5KHz clock (3/3); Good understanding shown on Mux/Demux (2/3); 7 segment decoder design shows clear understanding (2/3); Full block design for the adaptation shown. (4/6) Total (18/25)

18

ANALYSIS



Analysis of Xilinx Vivado simulation results: Good set of test results for the 6 bit device which have been analysed well (4/5). Good analysis of the 2 (not 3!) results shown in the report for the 8 bit device. (2/5) Analysis of adapted application results:

18

Very good set of results obtained which have been analysed well. 5KHz clock (4/5); Mux/Demux (4/5) and 7 segment decoder. (4/5) Total (18/25)

PRACTICAL APPLICATION AND DEP.



Downloading and testing to specification on Nexys 4 DDR FPGA board: 3 sets of test results are correct analysed well for the 8 bit Binary to BCD converter. (8/10) Downloading and testing of new application results on the Nexys 4 DDR FPGA board: Excellent use made of pictorial evidence in analysing test results obtained. (8/10) Total (16/20)

16

SKILLS FOR PROFESSIONAL PRAC.



Deployment of appropriate media: Satisfactory use made of screen dumps and good use made of pictorial evidence. (3/5) Report layout and readability: Report was constructed well and contained all aspects of assignment requirements. Good citation list noted. (3/5) Appropriate conclusion: Good general conclusion with some reflection noted (3/5) Total (9/15)

9