

Assignment 1 Joshua Beer

by Joshua Beer

FILE	12940_JOSHUA_BEER_ASSIGNMENT_1_JOSHUA_BEER_155114_409439 109.DOCX (7.37M)		
TIME SUBMITTED	18-JAN-2018 09:37AM (UTC+0000)	WORD COUNT	5092
SUBMISSION ID	81986944	CHARACTER COUNT	27122



VIRTUAL
INSTRUMENTATION
& CONTROL
SYSTEMS

Binary / BCD Conversion in VHDL

Joshua Beer

PSDE5029

Contents

.....	1
1. Introduction.....	4
2. Research	5
Structure of a typical VHDL file.	5
Binary to BCD conversion.....	8
IP Block Design in Vivado Design Suite.....	9
3. Design of Assignment Requirements.....	10
Binary to BCD 8 bit	10
4. Analysis of Xilinx Vivado Simulated Results	16
BIN to BCD 6 Bit Simulated Results	16
BIN to BCD 8 Bit Simulated Results	18
5. Analysis of Nexys4 FPGA Practical Results.....	21
6. Adapting Design to Suit New Purposes or Applications	24
7 segment display	24
Mux Demux.....	29
Five KHz clock	35
Final Design.....	39
Extra check for 7 segment display.	44
7. Analysis of Xilinx Vivado Simulated Results.....	46
Mux to Demux Simulation	46
7 Segment Display Simulation	49
FiveKHz Clock Simulation.....	52
8. Analysis of Nexys 4 DDR FPGA Practical Results	53
9. Conclusions.....	58
10. Appendices including bibliography	59
Bibliography	59

Figure 1 - Example of Entity	5
Figure 2 – Example of Entity Architecture Behavioural	6
Figure 3 - Binary to BCD Table	8
Figure 4 - BIN to BCD Design Source	10
Figure 5 – BIN to BCD Design Source (2)	11
Figure 6 – BIN to BCD Constraints	12
Figure 7 - BIN to BCD Simulation (1st Test)	13
Figure 8 - BIN to BCD Simulation (2nd and 3rd Test)	14
Figure 9 - Block Design Created Using the Above Sources	15
Figure 10 - Design Source 7 Seg	24
Figure 11 - Giving Seg Values for Numbers	25
Figure 12 - 7 Segment Display Example (wiki/Seven-segment_display)	25
Figure 13 - sa to sg/6 down to 0	26
Figure 14 - Constraints 7 Seg	26
Figure 15 - 7 Seg Simulation	27
Figure 16 - 7 Seg Simulation (2)	28
Figure 17 - Mux Demux Design Sources	29
Figure 18 - Mux Demux Process (Design Source)	31
Figure 19 - Mux Demux Simulation	32
Figure 20 - mux demux simulation (2)	33
Figure 21 - FiveKHz Clock Design Source	35
Figure 22 - FiveKHz Clock Simulation	37
Figure 23 - Final Design - Design Source	39
Figure 24 - Final Design - Design Source (2)	40
Figure 25 - Final Design - Design Source (3)	41
Figure 26 - Final Design Constraints	42
Figure 27 - Final Design Simulation	43
Figure 28 - Final Design Block Diagram	44
Figure 29 - 7 Segment SC Check Karnaugh Map and Boolean Expression	44
Figure 30 - Final Design IP Block (Appendix)	60

1. Introduction

The aims and objectives of this assignment will be to design and test the following VHDL files using the Xilinx Vivado Design Suite:

- 6 bit Binary to BCD converter.
- Extend the design to become an 8 bit Binary to BCD converter.
- Output and test on the specified FPGA using the IP Block Design in Vivado.
- Adapt the design to indicate the BCD value on 3 * 7 segment LED displays on the Nexys 4 DDR FPGA. This will involve the design of the following additional VHDL files: A 3*4 bit Mux_DeMux. 5 KHz clock from the internal 100MHz clock on the Nexys 4 DDR FPGA and BCD to 7 segment converter
- Output and test both designs on the specified FPGA using the IP Block Design in Vivado.

Very clear introduction

2. Research

Structure of a typical VHDL file.

The initial version of VHDL was designed to IEEE standard IEEE 1076-1987 (this is used to call upon any standard within this and use it in the VHDL as explained below), included a wide range of data types, including numerical (integer and real), logical (bit and Boolean), character and time, plus arrays of bit called bit_vector and of character called string.

The key advantage of VHDL, when used for systems design, is that it allows the behaviour of the required system to be described (modelled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

VHDL is now the recognised standard language along with Verilog and ABEL.

According to (3-basic-structure-of-a-vhdl-file/) "A digital system in VHDL consists of a design entity that can contain other entities that are then considered components of the top-level entity."

Once the entities are set up, each entity consists of a declaration and architecture body. The entity declaration contains the inputs and outputs while the architecture contains the description of what the entity is doing, for example in this assignment converting a 100 MHz inbuilt clock to a 5 KHz clock will be contained in the architecture of the entity.

```
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 FiveKHz_Clock is
6  Port ( CLK100MHZ : in STD_LOGIC;
7        Rst : in std_logic;
8        FiveKHz : out STD_LOGIC
9  );
10 end FiveKHz_Clock;
```

*Go
od
det
ail*

Figure 1 - Example of Entity

Figure 1 shows an example of the entity setting out the inputs and outputs of the FiveKHz clock.

```
11 architecture Behavioral of FiveKHz_Clock 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 end Behavioral;
```

Figure 2 – Example of Entity Architecture Behavioural

Figure 2 shows the architecture behavioural of the entity for the FiveKHz clock which is describing what will happen within the program process. This starts by declaring architecture behavioural of (insert specific program name) and ends by declaring end behavioural.

VHDL also uses pre-programed keywords to execute commands in the program code, these can come from pre-designated programs stored in the library and can be called upon by entering the correct IEEE keywords. *Good*

Alternatively keywords can be shorter and used to execute specific commands such as if, else, end and then etc. these can be seen in the example in figure 2.

The VHDL file can also use logic operators within its system and there is an example of this in task 7 of this assignment with the Boolean expression being formed for the 7 segment display. The program will simplify any known Karnaugh map and produce a Boolean expression which will use logic operators such as; and, or, nand, nor, xor, xnor and not to then produce the correct operation for the code introduced.

These can then be represented in physical form on a Nexys 4 FPGA board as in the assignment below (when referenced as just board).

Good detail given but lack of citations

Binary to BCD conversion.

Binary	BCD	Decimal
0000	0000 0000	0
0001	0000 0001	1
0010	0000 0010	2
0011	0000 0011	3
0100	0000 0100	4
0101	0000 0101	5
0110	0000 0110	6
0111	0000 0111	7
1000	0000 1000	8
1001	0000 1001	9
1010	0001 0000	10
1011	0001 0001	11
1100	0001 0010	12
1101	0001 0011	13
1110	0001 0100	14
1111	0001 0101	15

Figure 3 - Binary to BCD Table

Figure 3 shows the binary to BCD conversion table with its decimal. The decimal value of the binary and BCD number is shown on the right hand side of the table, in binary the value is shown as a summation of all the bits with the right bit being least significant. Taking 13 as an example 1101 will be $8 + 4 + 1 = 13$ whereas with BCD the bits are split into tens and units values. So using the example for 13 the units value in BCD is 0011 which uses the same principal as binary which is $2 + 1 = 3$, this is then placed as the unitary value alongside the tens value which in binary is 0001 which is 1 so 1 (tens) and 3 (units) is 13 in BCD. (author.asp?doc_id=1320492)

IP Block Design in Vivado Design Suite.

According to (XILINX, 2013) "As FPGAs become larger and more complex, and as design schedules become shorter, use of third party IP and design reuse is becoming mandatory. Xilinx recognizes the challenges designers face, and have created a powerful new feature within the Vivado™ Design Suite to assist you with these challenges."

This feature is called the Vivado IP integrator which allows the user to enter the Vivado design suite and combine different IP blocks already created to create a final simpler design. Designs are typically constructed at the interface level but may also be manipulated at the port level. An interface is a grouping of signals that share a common function.

Next, the graphical representation of this connection will be simple – a single connection (which can be seen in the assignment as the final design and simplified BIN to BCD 8 bit IP block designs). Finally, checks are done so that the program is aware of the specific interface that can be run to assure that all the required signals are connected properly.

According to (XILINX, 2013) "An advantage of the IP block is that the IP integrator implements dynamic, runtime DRCs to ensure, for example, that connections between the IP in an IP integrator design are compatible and that the IP themselves are properly configured."

This all enables the program to be simplified and easier to assemble for the user and program.

Good detail

(XILINX, 2013)

3. Design of Assignment Requirements

The following screenshots are of the initial design requirements to complete the first part of the assignment and to gain the relevant results. These designs will be explained below.

All of these screenshots will be written in VHDL and not Verilog. Unfortunately, the screenshots for the BIN to BCD 6 bit were overwritten when the 8 bit was made.

Binary to BCD 8 bit

Design Sources

```
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.VCOMPONENTS.ALL;
14 entity BIN_BCD_8_BIT_wrapper is
15 port (
16     B0 : in STD_LOGIC;
17     B1 : in STD_LOGIC;
18     B2 : in STD_LOGIC;
19     B3 : in STD_LOGIC;
20     B4 : in STD_LOGIC;
21     B5 : in STD_LOGIC;
22     B6 : in STD_LOGIC;
23     B7 : in STD_LOGIC;
24     H0 : out STD_LOGIC;
25     H1 : out STD_LOGIC;
26     SW15 : in STD_LOGIC;
27     T0 : out STD_LOGIC;
28     T1 : out STD_LOGIC;
29     T2 : out STD_LOGIC;
30     T3 : out STD_LOGIC;
31     Un0 : out STD_LOGIC;
32     Un1 : out STD_LOGIC;
33     Un2 : out STD_LOGIC;
34     Un3 : out STD_LOGIC
35 );
36 end BIN_BCD_8_BIT_wrapper;
37
38 architecture STRUCTURE of BIN_BCD_8_BIT_wrapper is
39     component BIN_BCD_8_BIT is
40     port (
41         B3 : in STD_LOGIC;
42         B4 : in STD_LOGIC;
43         B5 : in STD_LOGIC;
44         B6 : in STD_LOGIC;
45         B7 : in STD_LOGIC;
46         B0 : in STD_LOGIC;
47         Un0 : out STD_LOGIC;
48         Un1 : out STD_LOGIC;
49         Un2 : out STD_LOGIC;
50         Un3 : out STD_LOGIC;
51         T0 : out STD_LOGIC;
52         T1 : out STD_LOGIC;
53         T2 : out STD_LOGIC;
54         T3 : out STD_LOGIC;
55         H0 : out STD_LOGIC;
56         H1 : out STD_LOGIC;
57         SW15 : in STD_LOGIC;
58         B1 : in STD_LOGIC;
59         B2 : in STD_LOGIC
60     );
61 end component BIN_BCD_8_BIT;
```

*This is NOT the 6/8 bit
Binary to BCD design
file!*

Figure 4 - BIN to BCD Design Source

The figure above shows the standard logic inputs as follows:

B0, B1, B2, B3, B4, B5, B6 and B7 (binary inputs)

SW15 (switch 15 on the board)

The standard logic outputs are as follows:

Un0, Un1, Un2, Un3 (units)

T0, T1, T2, T3 (tens)

H0, H1 (hundreds)

These are all for the BCD part of the logic.

```
62 | begin
63 | BIN_BCD_8_BIT_i: component BIN_BCD_8_BIT
64 |     port map (
65 |         B0 => B0,
66 |         B1 => B1,
67 |         B2 => B2,
68 |         B3 => B3,
69 |         B4 => B4,
70 |         B5 => B5,
71 |         B6 => B6,
72 |         B7 => B7,
73 |         H0 => H0,
74 |         H1 => H1,
75 |         SW15 => SW15,
76 |         T0 => T0,
77 |         T1 => T1,
78 |         T2 => T2,
79 |         T3 => T3,
80 |         Un0 => Un0,
81 |         Un1 => Un1,
82 |         Un2 => Un2,
83 |         Un3 => Un3
84 |     );
85 | end STRUCTURE;
```

Figure 5 – BIN to BCD Design Source (2)

The figure above shows the values being assigned to their specific ports on the board.

Constraints

```
1  ##Switches
2
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_EMCCLK_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
14  ## LEDs
15
16 set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { Un0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
17 set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { Un1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
18 set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { Un2 }]; #IO_L17N_T2_A26_15 Sch=led[2]
19 set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { Un3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
20 set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { T0 }]; #IO_L7P_T1_D09_14 Sch=led[4]
21 set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { T1 }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
22 set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { T2 }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
23 set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { T3 }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
24 set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { H0 }]; #IO_L16N_T2_A15_DS1_14 Sch=led[8]
25 set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { H1 }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
```

Figure 6 – BIN to BCD Constraints

The figure above shows the switches for the binary values being assigned and for SW15 which will turn off the display board. The LEDs are assigned with the BCD values for units, tens and hundreds which will convert the binary value which is input and convert it into the BCD value displayed by the LEDs.

Simulation

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity BCD_BIN_8_tb is
4  -- Port ( );
5  end BCD_BIN_8_tb;
6  architecture Behavioral of BCD_BIN_8_tb is
7  -- Component Declaration for the Unit Under Test (UUT)
8  COMPONENT BIN_BCD_8_BIT_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  --Inputs
14  signal B0 : std_logic := '1';
15  signal B1 : std_logic := '1';
16  signal B2 : std_logic := '0';
17  signal B3 : std_logic := '1';
18  signal B4 : std_logic := '1';
19  signal B5 : std_logic := '0';
20  signal B6 : std_logic := '0';
21  signal B7 : std_logic := '0';
22  -- First test @ 27
23  signal SW15 : std_logic := '0';
24  --Outputs
25  signal Un0 : std_logic;
26  signal Un1 : std_logic;
27  signal Un2 : std_logic;
28  signal Un3 : std_logic;
29  signal T0 : std_logic;
30  signal T1 : std_logic;
31  signal T2 : std_logic;
32  signal T3 : std_logic;
33  signal H0 : std_logic;
34  signal H1 : std_logic;
35  --Clocks
36  -- No Clocks
37  begin
38  -- Instantiate the Unit Under Test (UUT)
39  uut: BIN_BCD_8_BIT_wrapper
40  PORT MAP (
```

Figure 7 - BIN to BCD Simulation (1st Test)

The figure above shows the test bench for the simulation with the same inputs and outputs as the design source. The binary value for the first test is then set as 27 being 00011011 and the SW15 set to 0 (off). This then applies these values for the simulation and allows them to be seen on the software which will be displayed and explained in the next task.

```

41 B0 => B0,
42 B1 => B1,
43 B2 => B2,
44 B3 => B3,
45 B4 => B4,
46 B5 => B5,
47 B6 => B6,
48 B7 => B7,
49 SW15 => SW15,
50 Un0 => Un0,
51 Un1 => Un1,
52 Un2 => Un2,
53 Un3 => Un3,
54 T0 => T0,
55 T1 => T1,
56 T2 => T2,
57 T3 => T3,
58 H0 => H0,
59 H1 => H1
60 );
61 -- Clock process definitions
62 -- No Clocks
63 -- Stimulus process
64 stim_proc: process
65 begin
66 wait for 1ms;
67 -- Second test @ 106
68 B0 <= '0';
69 B1 <= '1';
70 B2 <= '0';
71 B3 <= '1';
72 B4 <= '0';
73 B5 <= '1';
74 B6 <= '1';
75 B7 <= '0';
76 wait for 1ms;
77 -- Third test @ 156
78 B0 <= '0';
79 B1 <= '0';
80 B2 <= '1';
81 B3 <= '1';
82 B4 <= '1';
83 B5 <= '0';
84 B6 <= '0';
85 B7 <= '1';
86 wait;
87 end process;
88 end Behavioral;

```

Figure 8 - BIN to BCD Simulation (2nd and 3rd Test)

The figure above shows the 2nd and 3rd test values of 106 (01101010) and 156 (10011100) being set for the simulation which will be displayed and explained below.

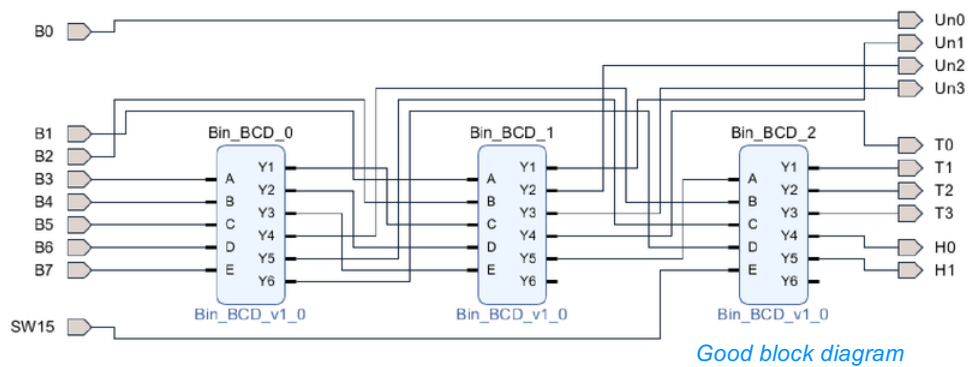


Figure 9 - Block Design Created Using the Above Sources

By creating HDL wrappers of the designs above the block diagram has been created which allows the BIN to BCD converter to be testing and the results to follow.

4. Analysis of Xilinx Vivado Simulated Results

By Simulating the BIN to BCD 6 bit and 8 bit converter the results below were obtained, these are then further explained below.

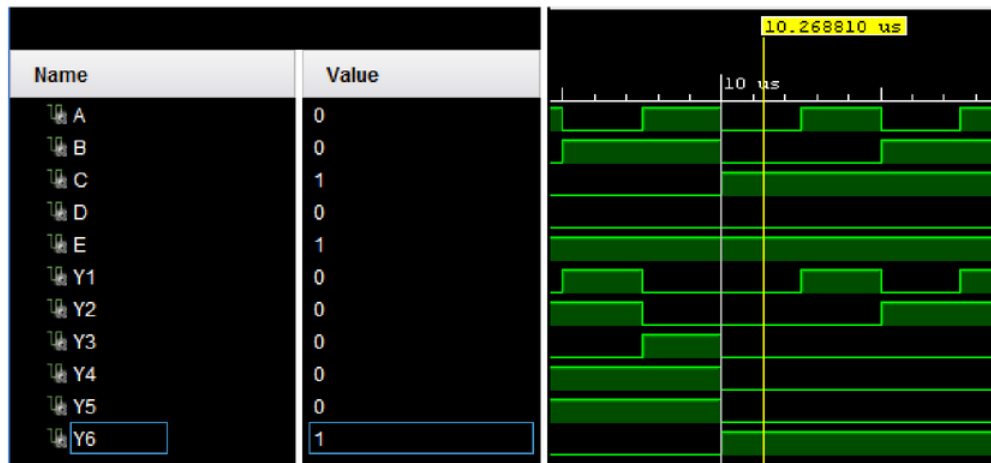
BIN to BCD 6 Bit Simulated Results



Numbers chosen = 17, 41 and 58

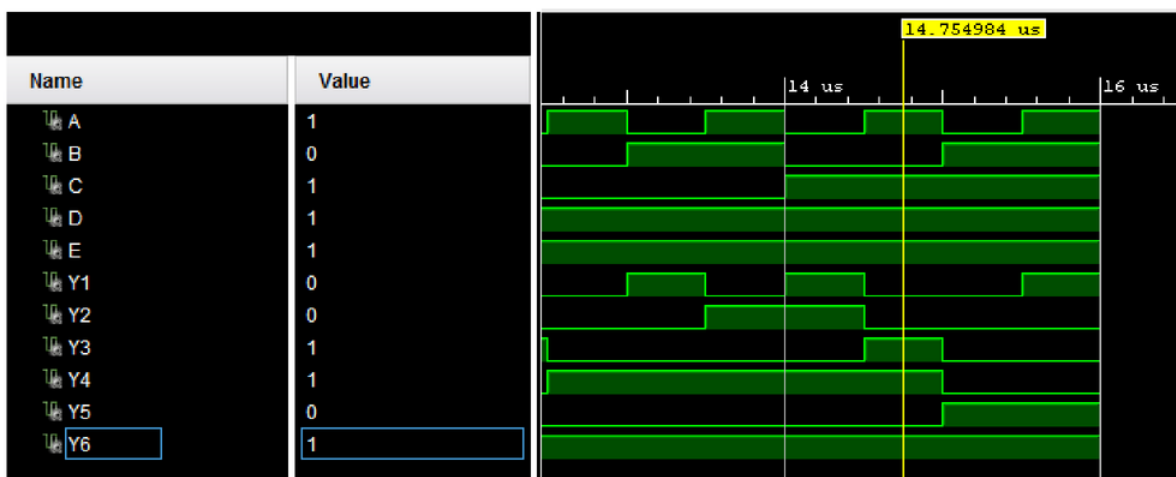
16/17	0	1	0	0	0	0/1	0	0	1	0	1	1	0/1
16/17	0	1	0	0	0	0/1	0	0	1	0	1	1	0/1

Results shown above for 17 which are 01000 (EDCBA) and 001011 (Y6, Y5, Y4, Y3, Y2, Y1)



40/41	1	0	1	0	0	0/1	1	0	0	0	0	0	0/1
-------	---	---	---	---	---	-----	---	---	---	---	---	---	-----

Results shown above for 41 which are 10100 (EDCBA) and 100000 (Y6, Y5, Y4, Y3, Y2, Y1)



58/59	1	1	1	0	1	0/1	1	0	1	1	0	0	0/1
-------	---	---	---	---	---	-----	---	---	---	---	---	---	-----

Results shown above for 58 which are 11101 (EDCBA) and 101100 (Y6, Y5, Y4, Y3, Y2, Y1)

The results for the BIN to BCD 6 bit converter are shown above, as with the table which is used as proof that the simulated results were a success and the binary (BIN) and BCD results match the chosen values.

Good set of results analysed correctly

BIN to BCD 8 Bit Simulated Results

Name	Value	0 ms
B0	1	
B1	1	
B2	0	
B3	1	
B4	1	
B5	0	
B6	0	
B7	0	
SW15	0	
Un0	1	
Un1	1	
Un2	1	
Un3	0	
T0	0	
T1	1	
T2	0	
T3	0	
H0	0	
H1	0	

Numbers chosen are 27, 106 and 156

```

signal B0 : std_logic := '1';
signal B1 : std_logic := '1';
signal B2 : std_logic := '0';
signal B3 : std_logic := '1';
signal B4 : std_logic := '1';
signal B5 : std_logic := '0';
signal B6 : std_logic := '0';
signal B7 : std_logic := '0';
-- First test @ 27

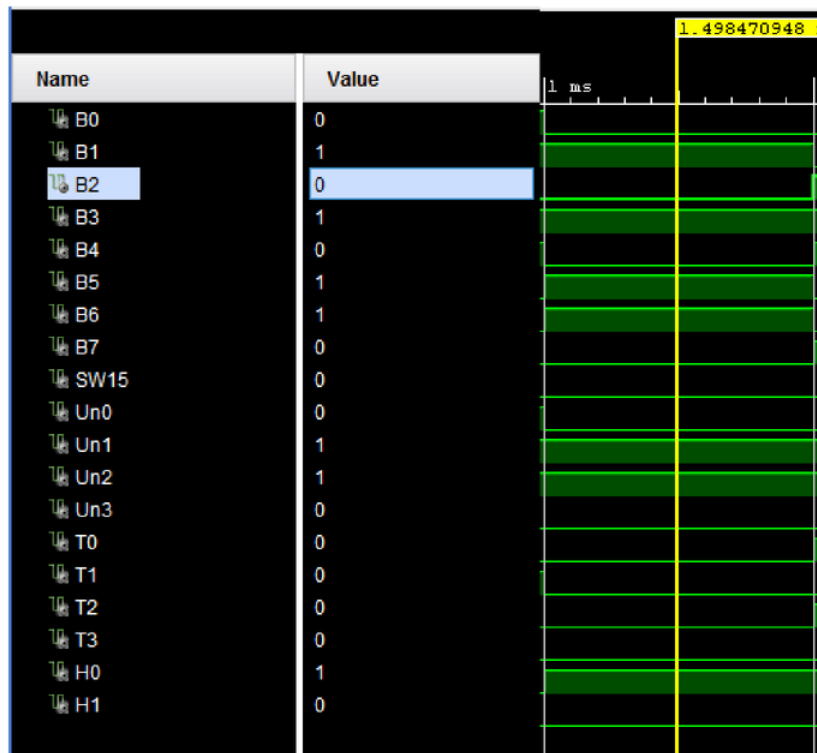
```

Results above show the value of 27 using 8 bit BIN and BCD values.

BIN value is 00011011 which is 27.

BCD value is units 0111 (7), tens 0010 (2) and hundreds 00 (0) therefore 027 or 27.

Good analysis



```

-- Second test @ 106
B0 <= '0';
B1 <= '1';
B2 <= '0';
B3 <= '1';
B4 <= '0';
B5 <= '1';
B6 <= '1';
B7 <= '0';

```

Results above show the value of 106 using 8 bit BIN and BCD values.

BIN value is 01101010 which is 106.

BCD value is units 0110 (6), tens 0000 (0) and hundreds 01 (1) therefore 106.

Good analysis

Name	Value	2 ms	2.507645260
B0	0		
B1	0		
B2	1		
B3	1		
B4	1		
B5	0		
B6	0		
B7	1		
SW15	0		
Un0	0		
Un1	1		
Un2	1		
Un3	0		
T0	1		
T1	0		
T2	1		
T3	0		
H0	1		
H1	0		

```
-- Third test @ 156
B0 <= '0';
B1 <= '0';
B2 <= '1';
B3 <= '1';
B4 <= '1';
B5 <= '0';
B6 <= '0';
B7 <= '1';
```

Results above show the value of 156 using 8 bit and bcd values.

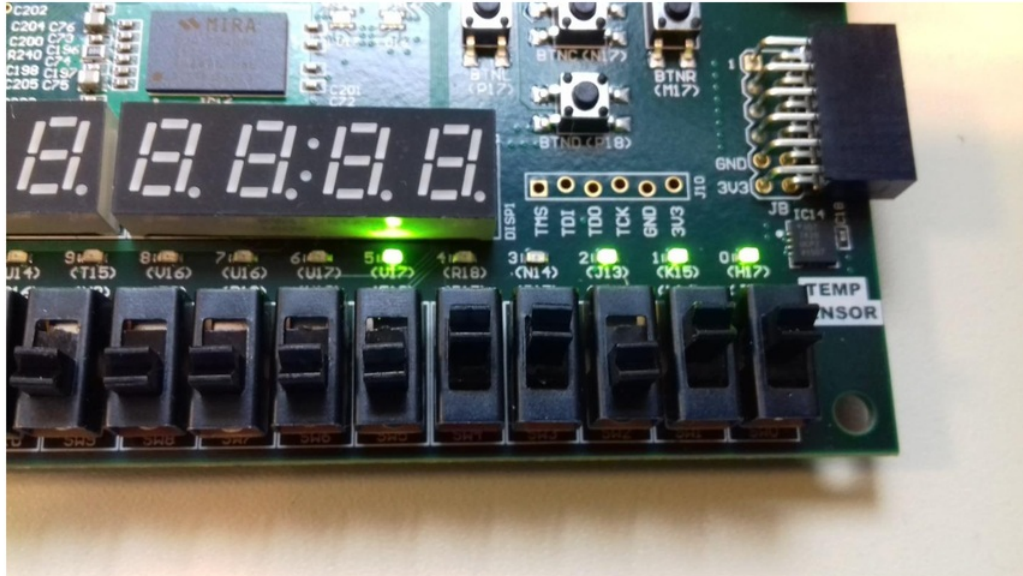
The top 'B' values are binary and therefore 10011100 is 156 in binary and matches both the program and simulated value.

The BCD value is then 01/0101/0110 which is 6 for unit values, 5 for tens value and 1 for hundred values, therefore 156 which is correct.

Good analysis

5. Analysis of Nexys4 FPGA Practical Results

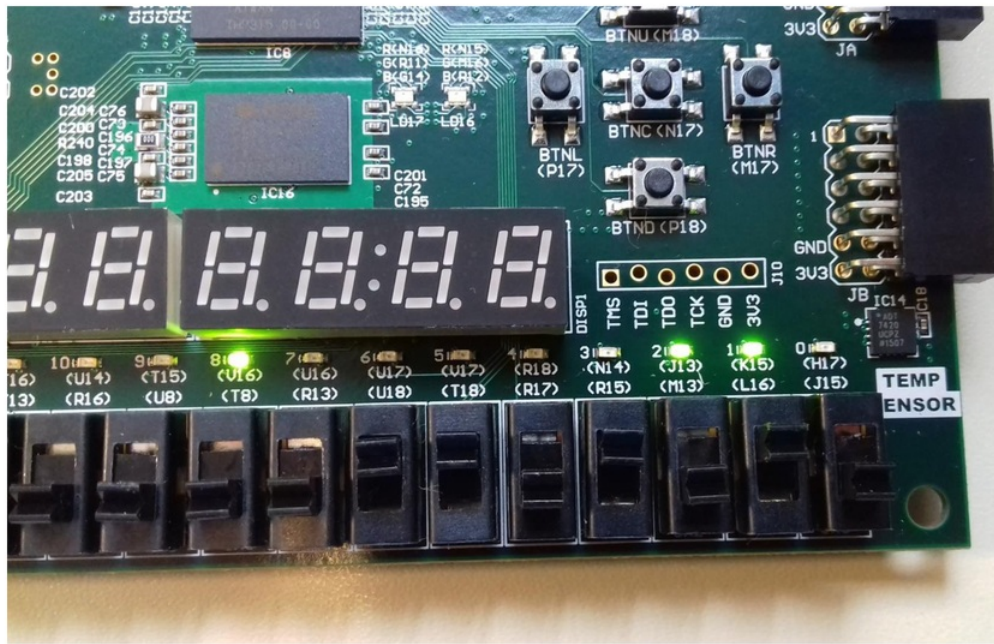
This part of the assignment will be to take the simulated results of the BIN to BCD 8 bit converter and compare them to practical results using the board.



The figure above shows the BIN to BCD of value 27. The figure shows the switches being 0000011011 which is binary for 27, while the LEDs are displaying units 0111 (7), tens 0010 (2) and hundreds 00 (0) which is 27 in BCD.

This is what is expected compared to the simulated result and is therefore a successful test.

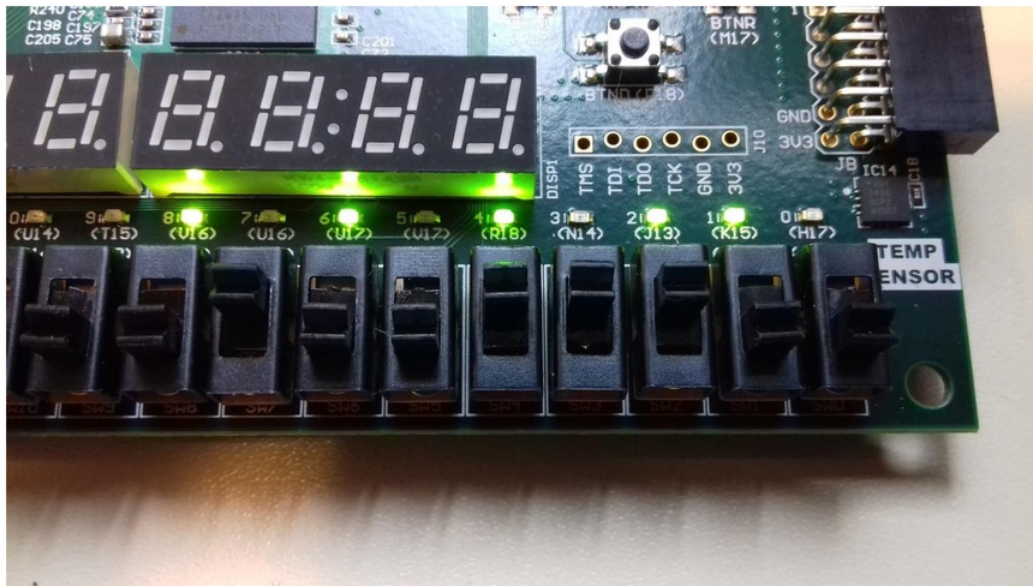
Good result analysed well



The figure above shows the BIN to BCD of value 106. The figure shows the switches being 0001101010 which is binary for 106, while the LEDs are displaying units 0110 (6), tens 0000 (0) and hundreds 01 (1) which is 106 in BCD.

This is what is expected compared to the simulated result and is therefore a successful test.

Again, good result shown and analysed correctly



The figure above shows the BIN to BCD of value 156. The figure shows the switches being 0010011100 which is binary for 156, while the LEDs are displaying units 0110 (6), tens 0101 (5) and hundreds 01 (1) which is 156 in BCD.

This is what is expected compared to the simulated result and is therefore a successful test.

As all three tests of the practical results for BIN to BCD 8 bit converter were successful it can be assumed that the design and download of the BIN to BCD converter was a success and that it can now be built upon in the next part of the assignment.

Good understanding shown through analysis of all test results

6. Adapting Design to Suit New Purposes or Applications

This part of the assignment will build on the BIN to BCD converter which was designed and tested above.

7 segment display

Design Sources

```
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
6  sa,sb,sc,sd,se,sf,sg : out std_logic
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
```

Figure 10 - Design Source 7 Seg

The figure above shows that the input standard logic is ports A, B, C and D with the output standard logic being sa, sb, sc, sd, se, sf and sg (which are the seven segments needed for the 7 segment display).

The next part sets out the signal values for Q and seg. Q being set as a value of 3 down to 0 with 3 being A and 0 being D. The values for seg is 6 down to 0 with sa being 6 and sg being 0.

```

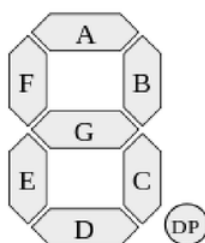
20 WHEN "0000" => -- 0
21 seg <= ("0000001");
22 WHEN "0001" => -- 1
23 seg <= ("1001111");
24 WHEN "0010" => -- 2
25 seg <= ("0010010");
26 WHEN "0011" => -- 3
27 seg <= ("0000110");
28 WHEN "0100" => -- 4
29 seg <= ("1001100");
30 WHEN "0101" => -- 5
31 seg <= ("0100100");
32 WHEN "0110" => -- 6
33 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;

```

Where does this information come from?

Figure 11 - Giving Seg Values for Numbers

Figure 11 shows that when the binary number is inputted then the corresponding value is output to the 7 segment display, an example is that 0000 is binary value 0 and this is output with 7 segment display of 0 with segment g turned off (7 segment display uses a reverse output to switch the segment on/off).



Good

Figure 12 - 7 Segment Display Example (wiki/Seven-segment_display)

Figure 12 shows that when seg sg is turned off the 7 segment display will show 0.

Figure 2 shows that when another value is entered then the screen will be blank and this ends the case and 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
52  end Behavioral;

```

Figure 13 - sa to sg/6 down to 0

The Figure above shows the sa to sg being assigned their values from 6 down to 0.

Constraints

```

1  ##Switches
2
3  set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { D }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
4  set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { C }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sv[1]
5  set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
6  set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { A }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
7  set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { SW0 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
8  set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { SW1 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
9  set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { SW2 }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
10 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { SW3 }]; #IO_L5N_T0_D07_14 Sch=sv[7]
11 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS10 } [get_ports { SW4 }]; #IO_L24N_T3_34 Sch=sv[8]
12 set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS10 } [get_ports { SW5 }]; #IO_25_34 Sch=sv[9]
13 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW6 }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sv[10]
14 set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW7 }]; #IO_L23P_T3_A03_D19_14 Sch=sv[11]
15
16 ##7 segment display
17
18 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { sa }]; #IO_L24N_T3_A00_D16_14 Sch=ca
19 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { sb }]; #IO_25_14 Sch=cb
20 set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { sc }]; #IO_25_15 Sch=cc
21 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { sd }]; #IO_L17P_T2_A26_15 Sch=cd
22 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { se }]; #IO_L19P_T2_MRCC_14 Sch=ce
23 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { sf }]; #IO_L19P_T3_A10_D26_14 Sch=cf
24 set_property -dict { PACKAGE_PIN L10 IOSTANDARD LVCMOS33 } [get_ports { sg }]; #IO_L4P_T0_D04_14 Sch=cg
25
26 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
27 set_property -dict { PACKAGE_PIN J10 IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
28 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
29 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T3_A22_15 Sch=an[3]
30 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L8N_T1_D12_14 Sch=an[4]
31 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
32 set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L23P_T3_35 Sch=an[6]
33 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
34

```

Figure 14 - Constraints 7 Seg

The figure above shows the design constraints for the 7 segment display and shows how they're assigned. The ports Q are set from A to D and will correspond to the values assigned in the design source, these are switches on the board and therefore let the user set the binary values which needs to be displayed. There is also switches 0 to 7 set for the BCD values later on.

The 7 segment display is then assigned its values from sa to sg which is the 6 down to 0 values explained above and A0 to A7 for the BCD values later on.

Simulation

```

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 sa : OUT std_logic;
17 sb : OUT std_logic;
18 sc : OUT std_logic;
19 sd : OUT std_logic;
20 se : OUT std_logic;
21 sf : OUT std_logic;
22 sg : OUT std_logic
23 );
24 END COMPONENT;
25 --Inputs
26 signal A : std_logic := '0';
27 signal B : std_logic := '0';
28 signal C : std_logic := '0';
29 signal D : std_logic := '0';
30 --Outputs
31 signal sa : std_logic;
32 signal sb : std_logic;
33 signal sc : std_logic;
34 signal sd : std_logic;
35 signal se : std_logic;
36 signal sf : std_logic;
37 signal sg : std_logic;
38 -- No clocks detected in port list. Replace <clock> below with
39 -- appropriate port name
40 constant A_period : time := 80 ns;
41 constant B_period : time := 40 ns;
42 constant C_period : time := 20 ns;
43 constant D_period : time := 10 ns;
44 BEGIN
45 -- Instantiate the Unit Under Test (UUT)

```

Good test bench shown

Figure 15 - 7 Seg Simulation

The Figure above shows the sources used to create the test bench and again setting the inputs as A, B, C and D with the outputs being sa, sb, sc, sd, se, sf and sg.

It then tells the display the clock time periods for each value A through to D with A being the longest at 80 ns and halving the value until it reached D at 10 ns.

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

Figure 16 - 7 Seg Simulation (2)

The figure above shows that the unit under test (UUT) is switching the values of A, B, C and D from 0 to 1 and this clock is pulsing through and looking for any change in value which in turn will change the value displayed on the 7 segment display. The values of pulses must differ in order for the clock to change on a rising or falling pulse and for the source to see this and differentiate between each signal.

Good detail

Mux Demux

Design Sources

```

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   process (cnt(1), cnt(0))
28   begin
29     if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
30     elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
31     elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
32     else Q <= "111";
33   end if;
34   end process;
35   AN0 <= Q(0);
36   AN1 <= Q(1);
37   AN2 <= Q(2);
38   process (Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1,Q(0),Q(1),Q(2))
39   begin
40     if (Q(0) = '0' and Un0 = '0') then D<='0';
41     elsif (Q(0) = '0' and Un0 = '1') then D<='1';
42     elsif (Q(1) = '0' and T0 = '0') then D<='0';
43     elsif (Q(1) = '0' and T0 = '1') then D<='1';
44     elsif (Q(2) = '0' and H0 = '0') then D<='0';
45     elsif (Q(2) = '0' and H0 = '1') then D<='1';
46   end if;
47   if (Q(0) = '0' and Un1 = '0') then C<='0';
48   elsif (Q(0) = '0' and Un1 = '1') then C<='1';
49   elsif (Q(1) = '0' and T1 = '0') then C<='0';
50   elsif (Q(1) = '0' and T1 = '1') then C<='1';
51   elsif (Q(2) = '0' and H1 = '0') then C<='0';
52   elsif (Q(2) = '0' and H1 = '1') then C<='1';
53   end if;

```

Figure 17 - Mux Demux Design Sources

The figure above shows the inputs for the mux demux design source as follows:

Rst (reset)

Un0, Un1, Un2, Un3 (units)

T0, T1, T2, T3 (tens)

H0, H1 (Hundreds)

And the outputs as follows:

AN0, AN1, AN2

A, B, C, D

The architecture of behavioural of the Mux_Demux is as follows:

Signal of the cnt (count) 1 down to 0.

Signal Q from 2 down to 0.

The process then begins calling on the fiveKHz timer (called from the explanation below) and the reset.

If reset is 1 then count is less than or equal to 11, if not then count will increment by 1 every time the fiveKHz clock is on a rising edge.

This process is ended and a new one for count (0) and count (1) begins as follows:

If count (1) = 0 and count (0) = 0 then Q is less than or equal to 110 otherwise;

If count (1) = 1 and count (0) = 0 then Q is less than or equal to 101 otherwise;

If count (1) = 0 and count (0) = 1 then Q is less than or equal to 011. If there is any other outcome other than the ones stated above (for example 1 and 1) then Q = 111.

AN0 is then assigned to Q0, AN1 to Q1 and AN2 to Q2. The process for this is looking to see what needs to be outputted to the board to display the value. Its looking for a value of Q 2 down to 0 and a value for each units, tens and hundreds, this is explained below.

Good detail

```

54 if (Q(0) = '0' and Un2 = '0') then B<='0';
55 elsif (Q(0) = '0' and Un2 = '1') then B<='1';
56 elsif (Q(1) = '0' and T2 = '0') then B<='0';
57 elsif (Q(1) = '0' and T2 = '1') then B<='1';
58 elsif (Q(2) = '0') then B<='0';
59 end if;
60 if (Q(0) = '0' and Un3 = '0') then A<='0';
61 elsif (Q(0) = '0' and Un3 = '1') then A<='1';
62 elsif (Q(1) = '0' and T3 = '0') then A<='0';
63 elsif (Q(1) = '0' and T3 = '1') then A<='1';
64 elsif (Q(2) = '0') then A<='0';
65 end if;
66 end process;
67 end Behavioral;

```

Figure 18 - Mux Demux Process (Design Source)

The figure above shows the same format as in figure 17, it multiplexes through each value of Q and its corresponding value for units, tens and hundreds. This is then output as a value for A, B, C and D with D being the least significant. It means that when outputting to the board the multiplexer will all share a value for D, C, B and A and for all except hundreds for B and A. This means a value for units will be displayed first which could be for example 0010 which would be 2 for units and then the same for tens and hundreds.

Not fully explained

Simulation

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Mux_Demux_tb is
4  -- Port ( ); No need to add info here as this is a testbench
5  end Mux_Demux_tb;
6  architecture Behavioral of Mux_Demux_tb is
7  component Mux_Demux -- Enter file to be tested
8  -- Copy the Port information from the VHDL file
9  Port ( FiveKHz : in STD_LOGIC;
10  Rst : in STD_LOGIC;
11  AN0,AN1,AN2 : out STD_LOGIC;
12  Un0,Un1,Un2,Un3 : in STD_LOGIC;
13  T0,T1,T2,T3 : in STD_LOGIC;
14  H0,H1 : in STD_LOGIC;
15  A,B,C,D : out STD_LOGIC
16  );
17  end component;
18  -- input signals
19  -- Example signal A : std_logic := '0';
20  signal FiveKHz : std_logic := '0';
21  signal Rst : std_logic := '1';
22  signal Un0 : std_logic := '1';
23  signal Un1 : std_logic := '0';
24  signal Un2 : std_logic := '0';
25  signal Un3 : std_logic := '0';
26  signal T0 : std_logic := '1';
27  signal T1 : std_logic := '0';
28  signal T2 : std_logic := '1';
29  signal T3 : std_logic := '0';
30  signal H0 : std_logic := '0';
31  signal H1 : std_logic := '1';
32  -- output signals
33  -- Example signal Q1 : std_logic := '0';
34  signal AN0 : std_logic;
35  signal AN1 : std_logic;
36  signal AN2 : std_logic;
37  signal A : std_logic;
38  signal B : std_logic;
39  signal C : std_logic;
40  signal D : std_logic;
41  -- clock period definitions
42  -- Example constant A_period : time := 8ns;
43  constant FiveKHz_period : time := 200us;
44  begin
45  -- Instantiate the UUT
46  UUT : Mux_Demux -- Enter file to be tested
47  Port Map (
```

Figure 19 - Mux Demux Simulation

The figure above shows the test bench for the mux demux where a simulated value of 251 is input using the binary values for units, tens and hundreds.

This then multiplexes through using the fiveKHz clock (200ms as explained below) and displays the resultant 251 value on the board.

```
48  -- Example A => A,
49  FiveKHz => FiveKHz,
50  Rst => Rst,
51  AN0 => AN0,
52  AN1 => AN1,
53  AN2 => AN2,
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  A => A,
65  B => B,
66  C => C,
67  D => D
68  );
69  -- clock process definitions
70  clk_process :process
71  begin
72    FiveKHz <= '0';
73    wait for FiveKHz_period/2; --for 5 ns signal is '0'.
74    FiveKHz <= '1';
75    wait for FiveKHz_period/2; --for next 5 ns signal is '1'.
76  end process;
77  -- Stimulus process
78  stim_proc: process
79  begin
80    wait for 1ms;
81    Rst <= '0';
82    wait for 5ms;
83    Rst <= '1';
84    wait;
85  end process;
86  end Behavioral;
```

Figure 20 - mux demux simulation (2)

The figure above shows the clock process for the mux demux process, the clock pulses every 5ns and every 10 ns is a complete cycle, therefore every 10ns the clock increments by 1 count.

Five KHz clock

Design Sources

```
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 FiveKHz_Clock is
6  Port ( CLK100MHZ : in STD_LOGIC;
7        Rst : in std_logic;
8        FiveKHz : out STD_LOGIC
9  );
10 end FiveKHz_Clock;
11 architecture Behavioral of FiveKHz_Clock 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 end Behavioral;
```

Figure 21 - FiveKHz Clock Design Source

The figure above shows that the FiveKHz clock uses the standard input from the 100MHz clock as its input port. It also has a reset (Rst) as another input. The output will be the FiveKHz clock.

The FiveKHz clock is made by having a count signal (cnt1) which is used as an integer which means the count must be a whole number, this is initially set to 0. Another signal is used temporarily (temp1) and this is initially set to 0 as well.

The process can now begin.

If reset is 1 (been pressed) then the temporary value and count value are set to 0 (reset).

On a rising edge of the 100 MHz clock and count is 10000 then temporary value is set to not the temporary value and the count set to 0. This basically means that every 10000 counts from the 100 MHz clock the counter resets.

If the number is anything but 10000 then the count will increment by 1. This then ends the process.

To work out the maths behind the FiveKHz clock there must be two pulse signals to complete a full cycle. Therefore 10000 counts must be completed twice at 100 MHz.

$$10000 \times 1/f$$

$$10000 \times 1/100 \text{ MHz} = 10000 \times 10 \times 10^{-9} = 100 \times 10^{-6}$$

This value is multiplied by two as explained above so $(100 \times 10^{-6}) \times 2 = 200 \times 10^{-6} = p$

$$\text{Frequency} = 1/p = 1/200 \times 10^{-6} = 5000$$

This is where the 5 KHz clock is coming from.

Very good detail shown

Simulation

```
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 FiveKHz_Clock_tb is
7  -- Port ( );
8  end FiveKHz_Clock_tb;
9  architecture Behavioral of FiveKHz_Clock_tb is
10 Component FiveKHz_Clock
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 --Outputs
20 signal FiveKHz : STD_LOGIC;
21 --Clock definitions
22 constant CLK100MHZ_period : time := 10ns;
23 begin
24 --Instantiate the UUT
25 UUT : FiveKHz_Clock
26 Port Map ( CLK100MHZ => CLK100MHZ,
27 Rst => Rst,
28 FiveKHz => FiveKHz
29 );
30 --Clock Process definitions
31 CLK100MHZ_process :process
32 begin
33 CLK100MHZ <= '0';
34 wait for CLK100MHZ_period/2;
35 CLK100MHZ <= '1';
36 wait for CLK100MHZ_period/2;
37 end process;
38 --Stimulus process definitions
39 stim_proc: process
40 begin
41 wait for 1ms;
42 Rst <= '0';
43 wait for 2ms;
44 Rst <= '1';
45 wait;
46 end process;
47 end Behavioral;
```

Figure 22 - FiveKHz Clock Simulation

The figure above shows that the simulation for the FiveKHz clock is using the time period of 10ns (1/100 MHz) which is then used in the UUT when the process begins.

This starts with the CLK100MHz being 0 and then waits for this to happen two times (period/2) and then changes to being a 1, this repeats again until it is back to 0 and will repeat until the simulation is satisfied.

The next part of the simulation process waits for 1 ms and then sets reset to 0 then waits for 2 ms and sets reset back to 1 which will reset the process and simulation.

Final Design

Design sources

```
10 | library IEEE;
11 | use IEEE.STD_LOGIC_1164.ALL;
12 | library UNISIM;
13 | use UNISIM.VCOMPONENTS.ALL;
14 | entity Seven_seg_wrapper is
15 |     port (
16 |         AN0 : out STD_LOGIC;
17 |         AN1 : out STD_LOGIC;
18 |         AN2 : out STD_LOGIC;
19 |         AN3 : out STD_LOGIC;
20 |         AN4 : out STD_LOGIC;
21 |         AN5 : out STD_LOGIC;
22 |         AN6 : out STD_LOGIC;
23 |         AN7 : out STD_LOGIC;
24 |         B0 : in STD_LOGIC;
25 |         B1 : in STD_LOGIC;
26 |         B2 : in STD_LOGIC;
27 |         B3 : in STD_LOGIC;
28 |         B4 : in STD_LOGIC;
29 |         B5 : in STD_LOGIC;
30 |         B6 : in STD_LOGIC;
31 |         B7 : in STD_LOGIC;
32 |         CLK100MHZ : in STD_LOGIC;
33 |         Rst : in STD_LOGIC;
34 |         SW14 : in STD_LOGIC;
35 |         SW15 : in STD_LOGIC;
36 |         sa : out STD_LOGIC;
37 |         sb : out STD_LOGIC;
38 |         sc : out STD_LOGIC;
39 |         sd : out STD_LOGIC;
40 |         se : out STD_LOGIC;
41 |         sf : out STD_LOGIC;
42 |         sg : out STD_LOGIC;
43 |     );
44 | end Seven_seg_wrapper;
```

Figure 23 - Final Design - Design Source

The figure above shows that the inputs are the binary values B0 to B7, the 100 MHz clock (inbuilt), the reset and switches 14 and 15.

The outputs are the 7 segments display sa to sg and AN0 to AN7.

```

46 architecture STRUCTURE of Seven_seg_wrapper is
47     component Seven_seg is
48     port (
49         CLK100MHZ : in STD_LOGIC;
50         Rst : in STD_LOGIC;
51         SW14 : in STD_LOGIC;
52         AN4 : out STD_LOGIC;
53         AN3 : out STD_LOGIC;
54         AN7 : out STD_LOGIC;
55         AN6 : out STD_LOGIC;
56         AN5 : out STD_LOGIC;
57         AN0 : out STD_LOGIC;
58         AN1 : out STD_LOGIC;
59         AN2 : out STD_LOGIC;
60         sa : out STD_LOGIC;
61         sb : out STD_LOGIC;
62         sc : out STD_LOGIC;
63         sd : out STD_LOGIC;
64         se : out STD_LOGIC;
65         sf : out STD_LOGIC;
66         sg : out STD_LOGIC;
67         B0 : in STD_LOGIC;
68         B1 : in STD_LOGIC;
69         B2 : in STD_LOGIC;
70         B3 : in STD_LOGIC;
71         B4 : in STD_LOGIC;
72         B5 : in STD_LOGIC;
73         B6 : in STD_LOGIC;
74         B7 : in STD_LOGIC;
75         SW15 : in STD_LOGIC
76     );
77 end component Seven_seg;

```

Figure 24 - Final Design - Design Source (2)

The figure above shows the architecture behind the 7 segment wrapper, this is the same as figure 20 and therefore needs no further explanation.

```

78   begin
79   Seven_seg_i: component Seven_seg
80       port map (
81           AN0 => AN0,
82           AN1 => AN1,
83           AN2 => AN2,
84           AN3 => AN3,
85           AN4 => AN4,
86           AN5 => AN5,
87           AN6 => AN6,
88           AN7 => AN7,
89           B0 => B0,
90           B1 => B1,
91           B2 => B2,
92           B3 => B3,
93           B4 => B4,
94           B5 => B5,
95           B6 => B6,
96           B7 => B7,
97           CLK100MHZ => CLK100MHZ,
98           Rst => Rst,
99           SW14 => SW14,
100          SW15 => SW15,
101          sa => sa,
102          sb => sb,
103          sc => sc,
104          sd => sd,
105          se => se,
106          sf => sf,
107          sg => sg
108      );
109  end STRUCTURE;

```

*Final design wrapper file shown
and analysed to a certain
extent*

Figure 25 - Final Design - Design Source (3)

The figure above shows the components in the 7 segment wrapper.

Constraints

```

1 Clock signal
2
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=sv[0]
9 set_property -dict { PACKAGE_PIN L16      IOSTANDARD LVCMOS33 } [get_ports { B1 }]; #IO_L3N_T0_DQS_ENMCLK_14 Sch=sv[1]
10 set_property -dict { PACKAGE_PIN M13      IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L6N_T0_D08_VREF_14 Sch=sv[2]
11 set_property -dict { PACKAGE_PIN R15      IOSTANDARD LVCMOS33 } [get_ports { B3 }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
12 set_property -dict { PACKAGE_PIN R17      IOSTANDARD LVCMOS33 } [get_ports { B4 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
13 set_property -dict { PACKAGE_PIN T18      IOSTANDARD LVCMOS33 } [get_ports { B5 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
14 set_property -dict { PACKAGE_PIN U18      IOSTANDARD LVCMOS33 } [get_ports { B6 }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
15 set_property -dict { PACKAGE_PIN R13      IOSTANDARD LVCMOS33 } [get_ports { B7 }]; #IO_L5N_T0_D07_14 Sch=sv[7]
16 #set_property -dict { PACKAGE_PIN T8      IOSTANDARD LVCMOS18 } [get_ports { H0 }]; #IO_L24N_T3_34 Sch=sv[8]
17 #set_property -dict { PACKAGE_PIN U8      IOSTANDARD LVCMOS18 } [get_ports { H1 }]; #IO_25_34 Sch=sv[9]
18 #set_property -dict { PACKAGE_PIN R16      IOSTANDARD LVCMOS33 } [get_ports { SW11 }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sv[10]
19 #set_property -dict { PACKAGE_PIN H6      IOSTANDARD LVCMOS33 } [get_ports { SW12 }]; #IO_L24P_T3_35 Sch=sv[12]
20 set_property -dict { PACKAGE_PIN U12      IOSTANDARD LVCMOS33 } [get_ports { Ret }]; #IO_L20P_T3_A08_D24_14 Sch=sv[13]
21 set_property -dict { PACKAGE_PIN U11      IOSTANDARD LVCMOS33 } [get_ports { SW14 }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sv[14]
22 set_property -dict { PACKAGE_PIN V10      IOSTANDARD LVCMOS33 } [get_ports { SW15 }]; #IO_L21P_T3_DQS_14 Sch=sv[15]
23
24 7 segment display
25
26 set_property -dict { PACKAGE_PIN T10      IOSTANDARD LVCMOS33 } [get_ports { sa }]; #IO_L24N_T3_A00_D16_14 Sch=ca
27 set_property -dict { PACKAGE_PIN R10      IOSTANDARD LVCMOS33 } [get_ports { sb }]; #IO_25_14 Sch=cb
28 set_property -dict { PACKAGE_PIN K16      IOSTANDARD LVCMOS33 } [get_ports { sc }]; #IO_25_15 Sch=cc
29 set_property -dict { PACKAGE_PIN K13      IOSTANDARD LVCMOS33 } [get_ports { sd }]; #IO_L17P_T2_A26_15 Sch=cd
30 set_property -dict { PACKAGE_PIN P15      IOSTANDARD LVCMOS33 } [get_ports { se }]; #IO_L13P_T2_MRCC_14 Sch=ce
31 set_property -dict { PACKAGE_PIN T11      IOSTANDARD LVCMOS33 } [get_ports { sf }]; #IO_L19P_T3_A10_D26_14 Sch=cf
32 set_property -dict { PACKAGE_PIN L18      IOSTANDARD LVCMOS33 } [get_ports { sg }]; #IO_L4P_T0_D04_14 Sch=cg
33
34 #set_property -dict { PACKAGE_PIN H15      IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
35
36 set_property -dict { PACKAGE_PIN J17      IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
37 set_property -dict { PACKAGE_PIN J16      IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
38 set_property -dict { PACKAGE_PIN T9       IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
39 set_property -dict { PACKAGE_PIN J14      IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T3_A22_15 Sch=an[3]
40 set_property -dict { PACKAGE_PIN P14      IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L6N_T1_D12_14 Sch=an[4]
41 set_property -dict { PACKAGE_PIN T14      IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
42 set_property -dict { PACKAGE_PIN K2       IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L23P_T3_35 Sch=an[6]
43 set_property -dict { PACKAGE_PIN U13      IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]

```

*Constraints file for final
design correct*

Figure 26 - Final Design Constraints

The figure above shows the constraints used for the final design, these are as follows:

Clock signal; this is using the internal 100 MHz clock to produce our FiveKHz clock which we have already made into a wrapper.

Switches; these are used for the binary number selection on the board along with the reset and SW14 and SW15 (ground).

7 segment display; this allows the binary values to be displayed on the 7 segment display if selected.

AN0 to AN7; this allows the wrappers using AN0 to AN7 to be used.

Simulation

```
10  library IEEE;
11  use IEEE.STD_LOGIC_1164.ALL;
12  library UNISIM;
13  use UNISIM.VCOMPONENTS.ALL;
14  entity Seven_seg_wrapper is
15  port (
16      AN0 : out STD_LOGIC;
17      AN1 : out STD_LOGIC;
18      AN2 : out STD_LOGIC;
19      AN3 : out STD_LOGIC;
20      AN4 : out STD_LOGIC;
21      AN5 : out STD_LOGIC;
22      AN6 : out STD_LOGIC;
23      AN7 : out STD_LOGIC;
24      B0 : in STD_LOGIC;
25      B1 : in STD_LOGIC;
26      B2 : in STD_LOGIC;
27      B3 : in STD_LOGIC;
28      B4 : in STD_LOGIC;
29      B5 : in STD_LOGIC;
30      B6 : in STD_LOGIC;
31      B7 : in STD_LOGIC;
32      CLK100MHZ : in STD_LOGIC;
33      Rst : in STD_LOGIC;
34      SW14 : in STD_LOGIC;
35      SW15 : in STD_LOGIC;
36      sa : out STD_LOGIC;
37      sb : out STD_LOGIC;
38      sc : out STD_LOGIC;
39      sd : out STD_LOGIC;
40      se : out STD_LOGIC;
41      sf : out STD_LOGIC;
42      sg : out STD_LOGIC
43  );
44  end Seven_seg_wrapper;
```

Figure 27 - Final Design Simulation

The figure above shows the simulation wrapper for the final design, this is setting the inputs and outputs which is the same as described in figure 20.

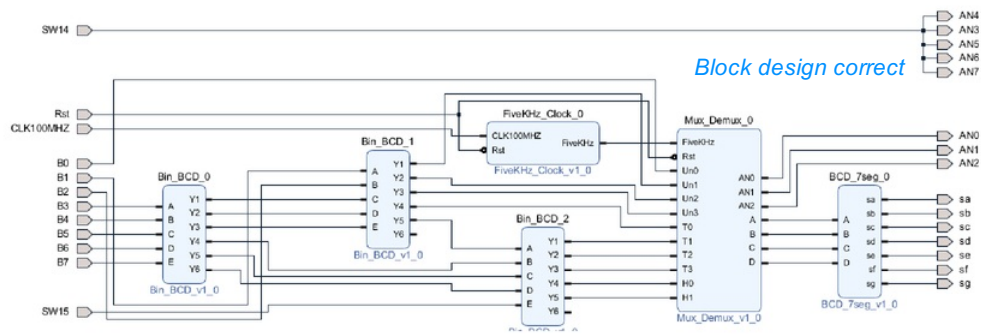


Figure 28 - Final Design Block Diagram

The figure above shows the final design block diagram which will be tested and the results shown below.

This block diagram is made by linking all the relevant wrappers as explained below which will enable the binary code selected on the switches to be displayed on the board.

Extra check for 7 segment display.

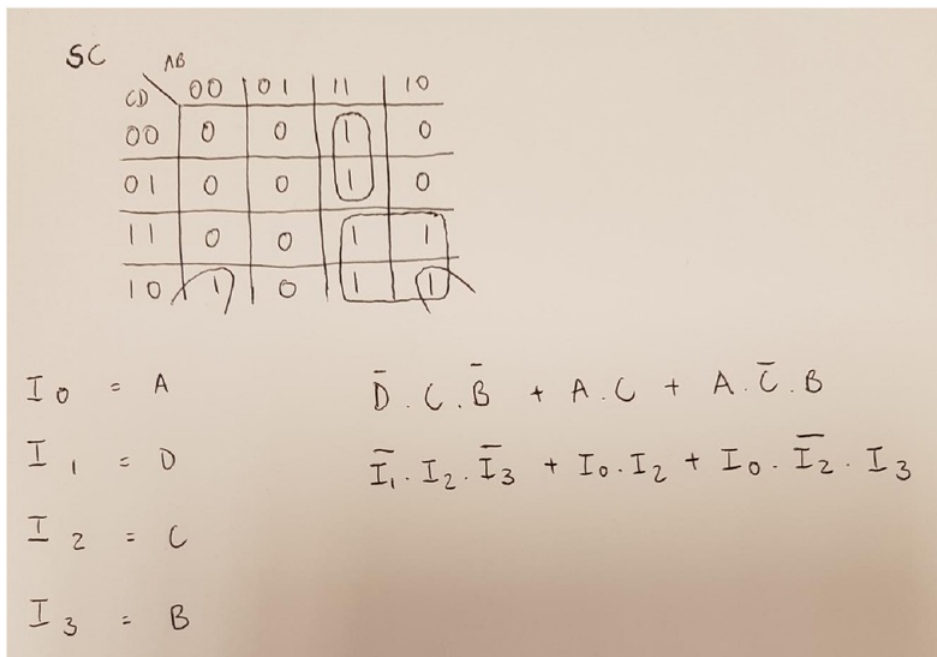


Figure 29 - 7 Segment SC Check Karnaugh Map and Boolean Expression

Good extra detail...

Figure 29 shows the Karnaugh map for the seven segment display values for sc, this is then grouped into 3 expressions which can then be simplified to the expression as shown in the figure.

This is then compared to the values assigned for A, B, C and D which the program refers to as I0, I1, I2 and I3. Which gives another expression which the program uses to produce the value for sc on the 7 segment display.

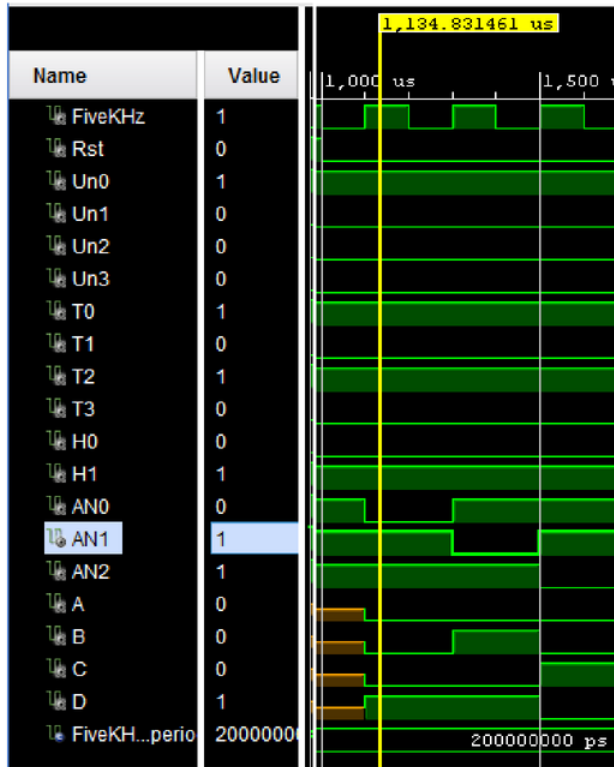
When trying to replicate this, there was problems as the program does not simplify the Karnaugh map down far enough as it uses a group of 2 where it could be simplified and be a group of 4 (vertically down).

Therefore the program could be designed better and work more efficiently if this was changed.

7. Analysis of Xilinx Vivado Simulated Results

By Simulating the Mux Demux, 7 segment display and FiveKHz clock the results below were obtained, these are then further explained below.

Mux to Demux Simulation



```

20    signal FiveKHz : std_logic := '0';
21    signal Rst : std_logic := '1';
22    signal Un0 : std_logic := '1';
23    signal Un1 : std_logic := '0';
24    signal Un2 : std_logic := '0';
25    signal Un3 : std_logic := '0';
26    signal T0 : std_logic := '1';
27    signal T1 : std_logic := '0';
28    signal T2 : std_logic := '1';
29    signal T3 : std_logic := '0';
30    signal H0 : std_logic := '0';
31    signal H1 : std_logic := '1';

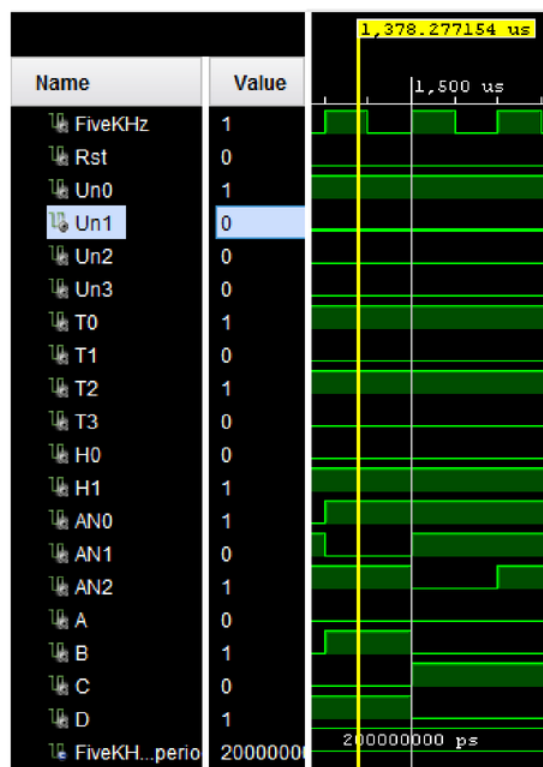
```

This is 251

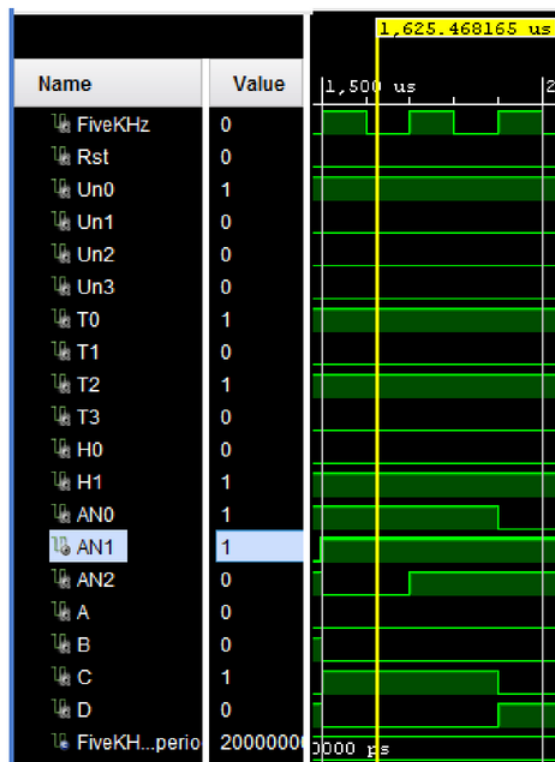
Units is 0001 (1), tens is 0101 (5) and hundreds is 10 (2) therefore 251 which matches the number chosen to be simulated.

This is proven in the test simulation above which reads 251 as well. *Very good*

This also works by ABCD, with D being least significant and left to right showing units, tens and hundreds. In the simulation above units show 1 for D which is 1 (units)



The value above shows D and B which is 1 and 4 so 5 (tens)



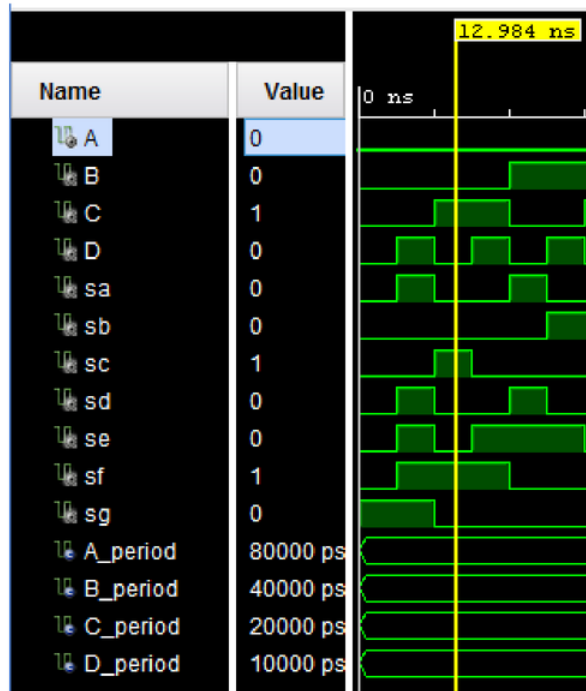
The simulation above shows 1 for C which is 2 (hundreds)

Putting all these together which is multiplexing totals 251.

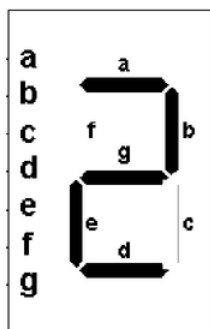
Multiplexing sends through the unit's value, then the tens and then hundreds and then puts them back together which makes the value. This can be seen through the simulation above.

Good set of test results and analysed well

7 Segment Display Simulation

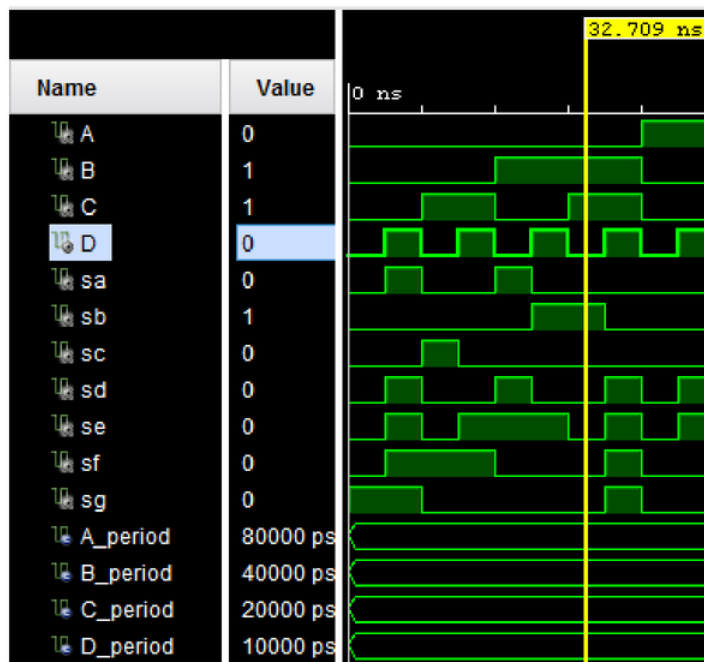


7 segment display value of 2 as C is 1 (as explained above with D being least significant and A the most). This also shows a binary code of 0100100 for GFEDCBA (where 0 is on and 1 off) so C and F are off and therefore the 7 segment display will show the value below.



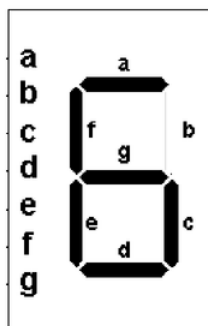
**7-Segment
LED Display**

Very good understanding shown



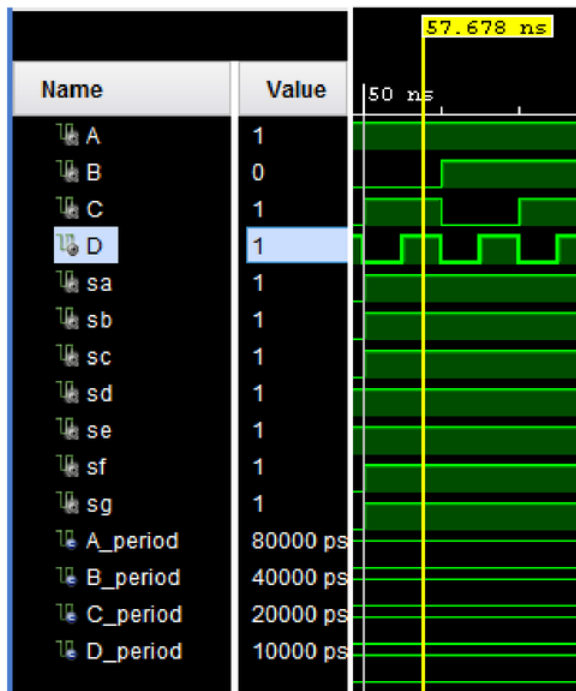
7 segment display value of 6 as B and C are 1 so $2+4 = 6$. This also shows a binary code of 00000110 for GFEDCBA (where 0 is on and 1 off)

So b is the only one off displaying 6 as shown below.



Again, good analysis

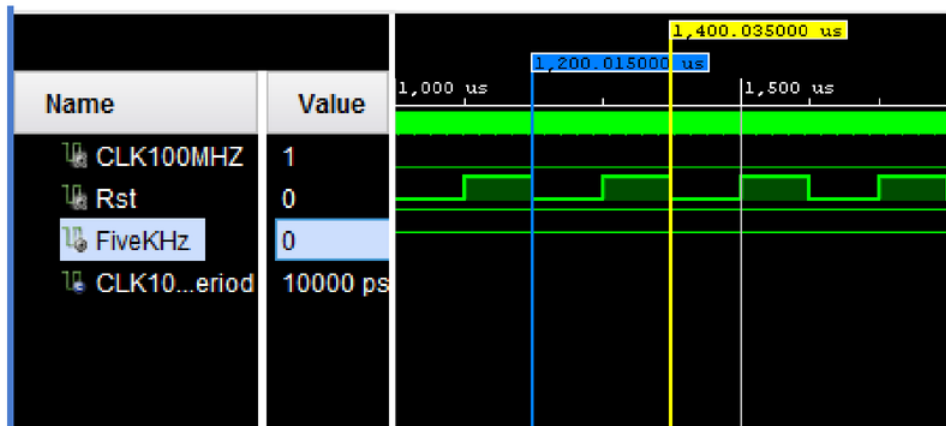
**7-Segment
LED Display**



7 segment display of 11 (D, C and A being 1+2+8) which is over 10 and therefore not displayed and all 7 segments are turned off with a binary code of 1111111.

Good detail...

FiveKHz Clock Simulation



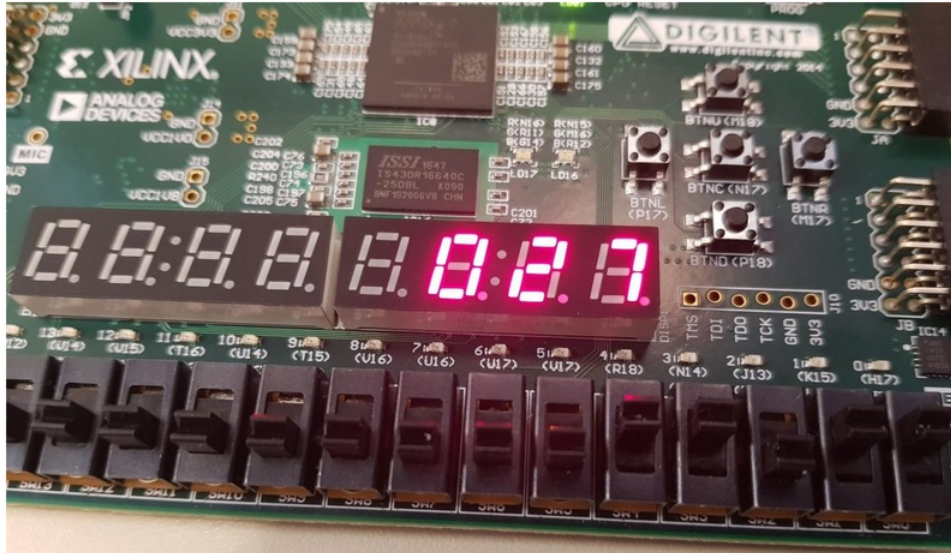
This shows the 200 us, which taken the reciprocal of this is 5 KHz clock pulse. This is simulated over 4 ms. Frequency = 1/period

So $1/200 \times 10^{-6} = 5000$

Good analysis

8. Analysis of Nexys 4 DDR FPGA Practical Results

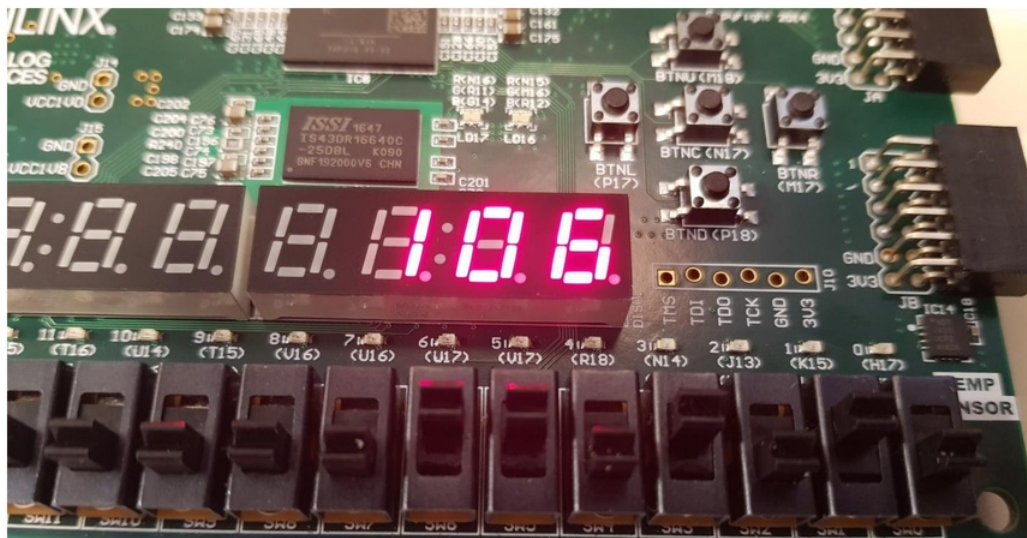
This part of the assignment will be to take the simulated results of the final design block diagram and compare them to practical results using the board.



The figure above shows the binary value entered for 27 being 00011011. This then displayed as 027 on the 7 segment display which matches the simulated results for the final design.

This is therefore a successfully test.

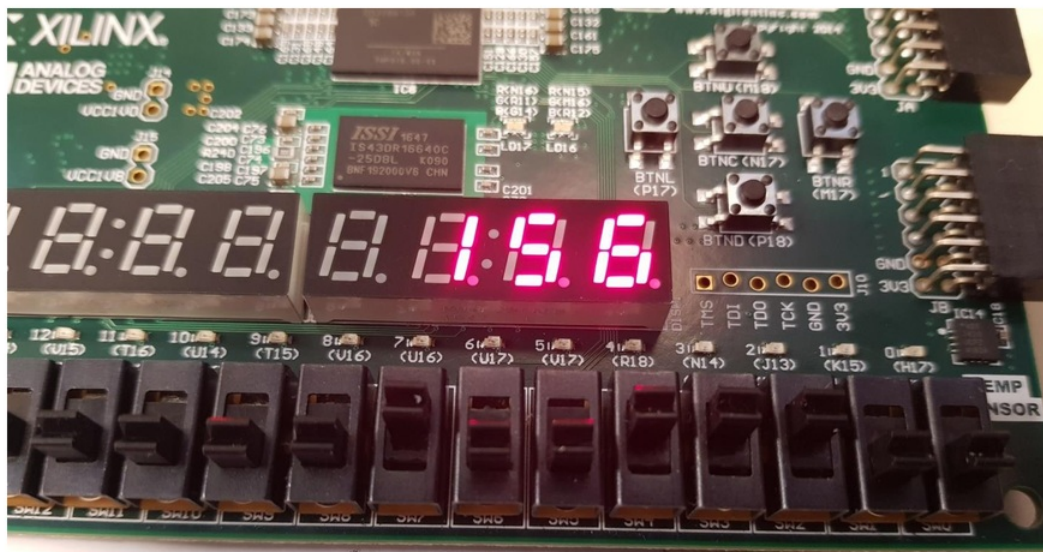
Good analysis of practical result



The figure above shows the binary value entered for 106 being 01101010. This then displayed as 106 on the 7 segment display which matches the simulated results for the final design.

This is therefore a successfully test.

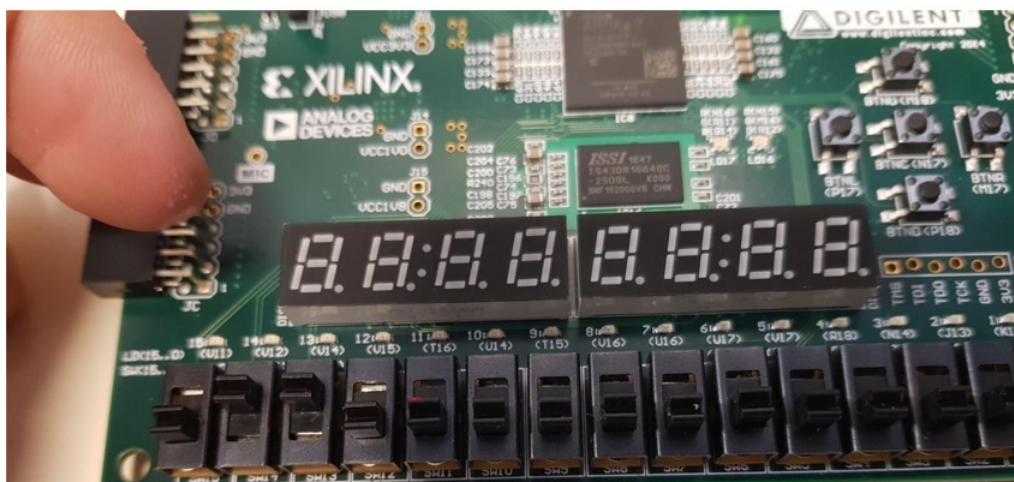
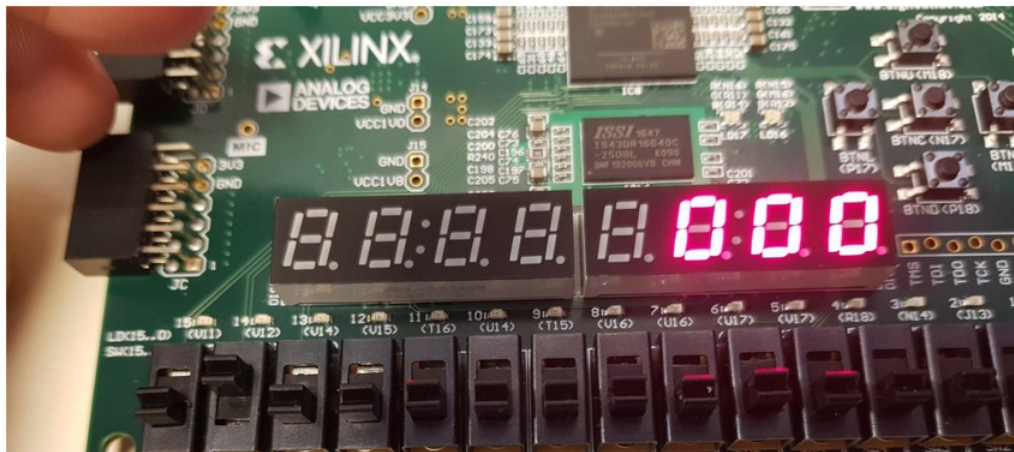
Again, good analysis shown



The figure above shows the binary value entered for 156 being 10011100. This then displayed as 156 on the 7 segment display which matches the simulated results for the final design.

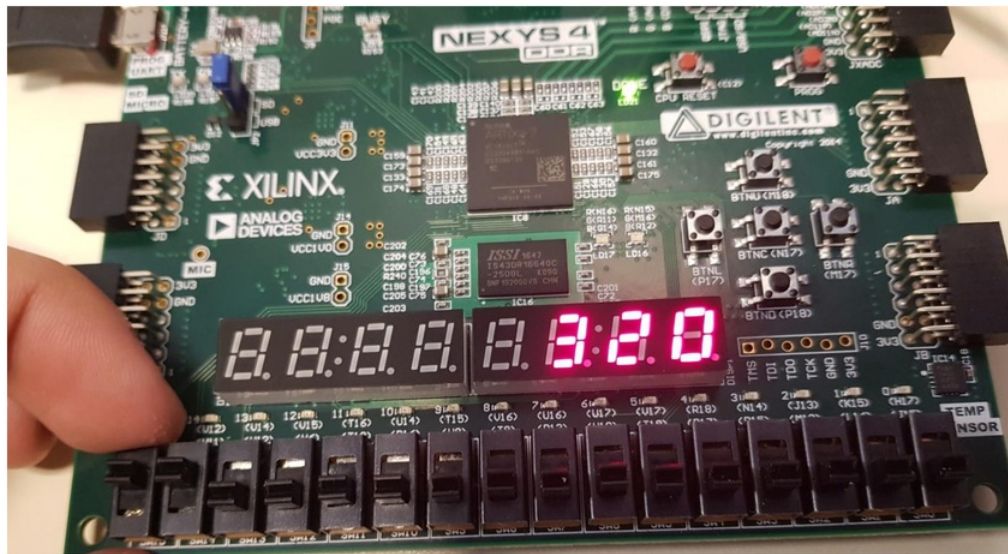
This is therefore a successfully test.

Good use made of pictorial evidence in the correct analysis of results obtained



The figures above are showing before and after pressing the reset button (SW13) with the binary value set to 0 for the test. Before the reset is switched the value 000 is displayed on the 7 segment display and once the reset is switched the display goes blank, just like the code says it will.

This is therefore a successfully test.



The figures above also show that when the SW15 is grounded and when switched off (0) it makes sure the system runs correctly, when switched (1) the system is not grounded and therefore does not work and adds 320 to any binary value set up.

This is an anomaly of the program and therefore is part of the test which does not affect the results of the simulated program but is interesting to show.

Excellent additional analysis

9. Conclusions

To conclude whether this assignment and program design has been a success, it will be beneficial to see if the aims and objectives of the assignment have been met.

The aims and objectives of this assignment were to be able to design and test the following VHDL files using the Xilinx Vivado Design Suite:

- 6 bit Binary to BCD converter.

This was successful as proved in the simulation of the 6 bit Binary to BCD converter

- Extend the design to become an 8 bit Binary to BCD converter.

This was successful as proved in the simulation and practical of the 8 bit Binary to BCD converter.

- Output and test on the specified FPGA using the IP Block Design in Vivado.

This again was successful with proof from the pictures gained from the practical using the FPGA board and the values used for the simulation and practical matched the results expected.

- Adapt the design to indicate the BCD value on 3 * 7 segment LED displays on the Nexys 4 DDR FPGA. This will involve the design of the following additional VHDL files: A 3*4 bit Mux_DeMux. 5 KHz clock from the internal 100MHz clock on the Nexys 4 DDR FPGA and BCD to 7 segment converter.

This was successful as the simulation results for the Mux Demux, 7 segments display and FiveKHz clock proved.

- Output and test both designs on the specified FPGA using the IP Block Design in Vivado.

This again was successful with proof from the pictures gained from the practical using the FPGA board and the values used for the simulation and practical matched the results expected.

Overall the assignment can therefore be considered a success as all the aims and objectives were successfully met.

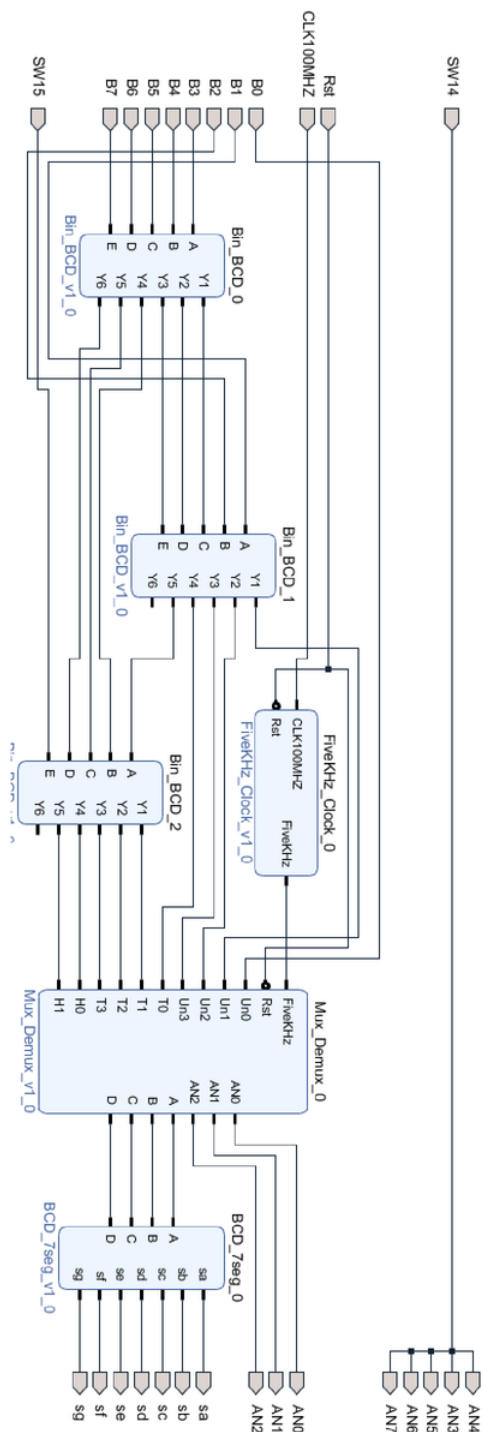
Good general conclusion given which clearly stated all assignment objectives were met

10. Appendices including bibliography

Bibliography

- 3-basic-structure-of-a-vhdl-file/*. (n.d.). Retrieved January 2018, from
www.euroelectronica.ro: <http://www.euroelectronica.ro/3-basic-structure-of-a-vhdl-file/>
- author.asp?doc_id=1320492*. (n.d.). Retrieved January 2018, from
www.eetimes.com: https://www.eetimes.com/author.asp?doc_id=1320492
- wiki/Seven-segment_display*. (n.d.). Retrieved January 2018, from
<https://en.wikipedia.org>: https://en.wikipedia.org/wiki/Seven-segment_display
- XILINX. (2013, October 2nd).
/support/documentation/sw_manuals/xilinx2013_3/ug994-vivado-ip-subsystems.pdf. Retrieved January 2018, from www.xilinx.com:
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug994-vivado-ip-subsystems.pdf

Citations could have been more extensive



Good block diagram
shown for the final
adapted design

Figure 30 - Final Design IP Block (Appendix)

Assignment 1 Joshua Beer

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

63/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

Text Comment. Very clear introduction

PAGE 5

Text Comment. Good detail

PAGE 6

Text Comment. Good

PAGE 7

Text Comment. Good detail given but lack of citations

PAGE 8

PAGE 9

Text Comment. Good detail

PAGE 10

Text Comment. This is NOT the 6/8 bit Binary to BCD design file!

PAGE 11

PAGE 12

PAGE 13

Text Comment. Good block diagram

Text Comment. Good set of results analysed correctly

Text Comment. Good analysis

Text Comment. Good analysis

Text Comment. Good analysis

Text Comment. Good result analysed well

Text Comment. Again, good result shown and analysed correctly

Text Comment. Good understanding shown through analysis of all test results

Text Comment. Where does this information come from?

Text Comment. Good

Text Comment. Good test bench shown

PAGE 28

PAGE 29

Text Comment. Good detail

PAGE 30

Text Comment. Good detail

PAGE 31

Text Comment. Not fully explained

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

Text Comment. Very good detail shown

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

Text Comment. Final design wrapper file shown and analysed to a certain extent

PAGE 42

Text Comment. Constraints file for final design correct

PAGE 43

PAGE 44

Text Comment. Block design correct

Text Comment. Good extra detail...

Text Comment. Very good

Text Comment. Good set of test results and analysed well

Text Comment. Very good understanding shown

Text Comment. Again, good analysis

Text Comment. Good detail...

Text Comment. Good analysis

Text Comment. Good analysis of practical result

Text Comment. Again, good analysis shown

Text Comment. Good use made of pictorial evidence in the correct analysis of results obtained

Text Comment. Excellent additional analysis

Text Comment. Good general conclusion given which clearly stated all assignment objectives were met

PAGE 59

Text Comment. Citations could have been more extensive

PAGE 60

Text Comment. Good block diagram shown for the final adapted design

GRADING FORM: CSG-RUBIC

JOSHUA BEER

63

RESEARCH-INFORMED LITERATURE



Structure of a typical VHDL file: Good detail shown but no research referenced. (2/5) Binary to BCD conversion: Good detail shown. (3/5) IP Block Design in Vivado: Good information given. (3/5) Total (8/15)

8

KNOWLEDGE AND UNDERSTANDING



Design of 6/8 bit Binary to BCD Converter: 6 bit design not shown in report (1/5); 8 bit converter block diagram design shown in report with appropriate test bench. (3/5). 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 (15/25)

15

ANALYSIS

- | | | |
|---|---|-----------|
|  | Analysis of Xilinx Vivado simulation results: 6 bit design simulated and analysed well (3/5); 8 bit design analysed well showing students understanding of the simulated results obtained. (4/5). Analysis of adapted application results: Satisfactory set of results obtained which have been analysed well. 5KHz clock (4/5); Mux/Demux (2/5) and 7 segment decoder. (3/5) Total (16/25) | 16 |
|---|---|-----------|

PRACTICAL APPLICATION AND DEP.

- | | | |
|---|--|-----------|
|  | Downloading and testing to specification on Nexys 4 DDR FPGA board: Good understanding shown through analysis of all practical test results for the 8 bit device. (7/10) Downloading and testing of new application results on the Nexys 4 DDR FPGA board: Good use made of pictorial evidence in the correct analysis of results obtained. (8/10) Total (15/20) | 15 |
|---|--|-----------|

SKILLS FOR PROFESSIONAL PRAC.

- | | | |
|---|--|----------|
|  | Deployment of appropriate media: Good use made of screen dumps and pictorial evidence in design and analysis of results. (3/5) Report layout and readability: 6/8 bit design not well presented but adapted design and analysis shown clearly. (3/5) Appropriate conclusion: Good general conclusion given which clearly stated all assignment objectives were met. (3/5) Total (9/15) | 9 |
|---|--|----------|