

# PSDE 5029 Final

*by* Nathan Smith

---

|                |                                                                    |                 |       |
|----------------|--------------------------------------------------------------------|-----------------|-------|
| FILE           | 15460_NATHAN_SMITH_PSDE_5029_FINAL_155114_536361257.PDF<br>(1.91M) |                 |       |
| TIME SUBMITTED | 18-JAN-2018 09:43AM (UTC+0000)                                     | WORD COUNT      | 9090  |
| SUBMISSION ID  | 81987312                                                           | CHARACTER COUNT | 46448 |



# HND ELECTRICAL & ELECTRONIC ENGINEERING

## PSDE 5029 VIRTUAL INSTRUMENTATION & CONTROL

### ASSIGNMENT 1 BINARY/ BCD CONVERSION IN VHDL

Nathan Smith

Word Count = 8843

## Contents

|                                                               |    |
|---------------------------------------------------------------|----|
| 1. Introduction .....                                         | 5  |
| 2. Research .....                                             | 6  |
| 2.1 Structure of a typical VHDL file. ....                    | 6  |
| 2.2 Binary to BCD conversion. ....                            | 10 |
| 2.3 IP Block Design in the Vivado Design Suite. ....          | 12 |
| 3. Design of assignment requirements .....                    | 13 |
| 4. Analysis of Xilinx Vivado simulated results .....          | 22 |
| 5. Analysis of Nexys4 FPGA practical results .....            | 28 |
| 6. Adapting design to suit new purposes or applications ..... | 31 |
| .....                                                         | 32 |
| 7. Analysis of Xilinx Vivado simulated results .....          | 43 |
| 8. Analysis of Nexys 4 DDR FPGA practical results .....       | 49 |
| 9. Conclusions .....                                          | 52 |
| 10. Appendices including bibliography .....                   | 53 |
| Works Cited .....                                             | 53 |

|                                                                          |    |
|--------------------------------------------------------------------------|----|
| Figure 1: Behavioural, Structural and Physical VHDL (Spiegel, 2001)..... | 6  |
| Figure 2: Entity example.....                                            | 7  |
| Figure 3: Architecture of system.....                                    | 8  |
| Figure 4: 4 Bit Number to Binary Values .....                            | 10 |
| Figure 5: Truth Table Binary - BCD (Larson, 2017).....                   | 11 |
| Figure 6: IP Block Design Binary - BCD .....                             | 12 |
| Figure 7: 6bit Binary to bcd design file .....                           | 14 |
| Figure 8: Design Source.....                                             | 14 |
| Figure 9: Library Commands.....                                          | 14 |
| Figure 10: Entity 6bit .....                                             | 15 |
| Figure 11: 6 Bit Architecture start.....                                 | 15 |
| Figure 11: 6 Bit Architecture End .....                                  | 15 |
| Figure 13: 6bit to BCD TB 1 .....                                        | 16 |
| Figure 14: 6 Bit to BCD TB 2.....                                        | 16 |
| Figure 15: Test Bench entity .....                                       | 17 |
| Figure 16: Inputs, Outputs and Clock 6bit .....                          | 17 |
| Figure 17: 6bit Unit Under Test .....                                    | 18 |
| Figure 18: 8-bit binary to BCD convertor .....                           | 19 |
| Figure 19: 8bit Wrapper File.....                                        | 19 |
| Figure 20: 8bit Constraints .....                                        | 20 |
| Figure 21: 8bit Simulation, first test .....                             | 21 |
| Figure 22: 8bit simulation, second and third test .....                  | 21 |
| Figure 23: 6bit 22/23 simulation .....                                   | 22 |
| Figure 24: 6bit 22/23 enhanced.....                                      | 22 |

|                                           |    |
|-------------------------------------------|----|
| Figure 25: 6 bit 30/31 Simulation.....    | 23 |
| Figure 26: 30/31 enhanced .....           | 23 |
| Figure 27: 6 bit 8/9 Sim .....            | 24 |
| Figure 28: 8/9 enhanced .....             | 24 |
| Figure 29: 8 bit 232 Sim .....            | 25 |
| Figure 30: 232 Table .....                | 25 |
| Figure 31: 8 bit, 75 Sim .....            | 26 |
| Figure 32: 75 Sim enhanced .....          | 26 |
| Figure 33: 8 bit, 217 Simulation.....     | 27 |
| Figure 34: 217 enhanced .....             | 27 |
| Figure 35: Nexys 4 board Labelled.....    | 28 |
| Figure 36: 8-bit test 232 .....           | 29 |
| Figure 37: 8-bit test 75 .....            | 29 |
| Figure 38: Test 8 bit 217 .....           | 30 |
| Figure 39: 5khz clock .....               | 31 |
| Figure 40: 5 kHz clock test bench .....   | 32 |
| Figure 41: Mux Demux VHDL .....           | 33 |
| Figure 42: MD VHDL 2.....                 | 34 |
| Figure 43: MD simulation file.....        | 35 |
| Figure 44: DM test bench 2.....           | 36 |
| Figure 45: 7 Segment display VHDL 1 ..... | 37 |
| Figure 46: 7 segment Simulation.....      | 39 |
| Figure 47: 7 Seg Sim 2.....               | 40 |
| Figure 48: 7 Segment Constraints.....     | 41 |

|                                                            |    |
|------------------------------------------------------------|----|
| Figure 49: I/P block design for adaptation for 7 seg ..... | 42 |
| Figure 50: Simulation of 5khz clock.....                   | 43 |
| Figure 51: Mux_demux Sim .....                             | 44 |
| Figure 52: Value of 3 .....                                | 44 |
| Figure 53: Mux_demux tens.....                             | 45 |
| Figure 54: Table value for tens.....                       | 45 |
| Figure 55: Mux_Demux hundreds .....                        | 46 |
| Figure 56: Hundreds mux_demux .....                        | 46 |
| Figure 57: 7 Seg Sim value 1 .....                         | 47 |
| Figure 58: 7 seg value of 3 .....                          | 47 |
| Figure 59: 7 Seg Value of 5.....                           | 48 |
| Figure 60: Practical 7 seg 232.....                        | 49 |
| Figure 61: 232 Value sim table.....                        | 49 |
| Figure 62: Simulation for 075 .....                        | 50 |
| Figure 63: Table for 075 Sim.....                          | 50 |
| Figure 64: 7 seg practical for 217 .....                   | 51 |
| Figure 65: Simulation table for the value of 217 .....     | 51 |

## 1. Introduction

This assignment will include the use of the Vivado design suite along with the Nexys 4 FPGA boards to design simulate then test a series of designs for a control process. The first part of the assignment will look into research of a typical vhdI file, the task will explain what a VHDL is and how it is contracted. The information has been researched from creditable sources and cross checked against other sources for accuracy, the research was accompanied by a number of images further explaining text used to show a better understanding.

The next part looks at the Binary to BCD conversion, this part of the research again used creditable sources which were cross referenced for accuracy to give an explanation of the two different types of codes mentioned in the questions and why they are converted. This part was accompanied by images again to further explain points of research. The last part of the research looked at the IP block design used in the assignment, the source of research was the direct company who creates this technology so the research was creditable, the block design is explained along with an illustration of the assignment block design.

The following tasks looked at using the software to build a variety of designs then simulate and test these designs, the designs are then explained along with explanations of what the code is doing in the designs and analysis of the simulations and the practical results. The simulation results are explained along with screenshots taken from using the system and the testing is explained with the aid of photos taken on a 12mp camera to show the practical application of the designs.

*to be*

The assignment is finished with a conclusion explained what was learnt from the assignment, if the assignment and tasks met the expected results and if any problems were faced an explanation of this. The assignment includes a contents page above, a table of figures, and a bibliography with all citations included.

*Good, clear introduction given*

## 2. Research

### 2.1 Structure of a typical VHDL file.

A VHDL file is an abbreviation for a 'Very High Speed Integrated Circuit', these circuits were designed to and commissioned by primarily the United States government to create these integrated circuits that were high speed due to no systems such existing. After their development, these VHDL's have become a standard language in hardware descriptions along with other hardware's such as 'Verilog' and 'ABEL'. A VHDL system has a number of different ways of abstracting its information, the system will have a 'Structural' form which will contain a series of component representation like logic gates and connection lines representing inputs and outputs. The structural form is shown below in figure 1.

The next form of representation is the 'Behavioural' form which will consist of a series of code to describe how the system will behave in terms of inputs and outputs and any commands. The behavioural form is also shown in figure 1 below. The last form of representation would be the 'Physical' representation, this would be the application of the deigned system onto a physical device such as a Nexys4 FPGA board as used in the assignment. This is also represented below in figure 1.

(Spiegel, 2001)

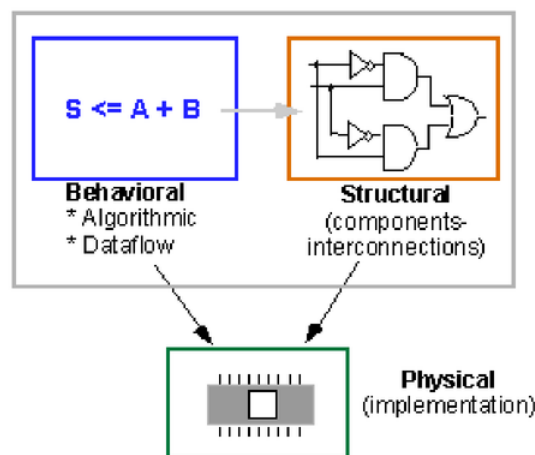


Figure 1: Behavioural, Structural and Physical VHDL (Spiegel, 2001)

The structure of the VHDL file is set up around an 'Entity', this is the main body of the system and can include additional entities called components of the main entity. The entity is controlled or designed around a declaration of this entity along with a body of the system call the 'architecture'. The declaration of this entity is where the input/output signals are defined and the 'architecture' will contain the system components, processing signals, entities which are interconnected and the overall description for the entity.

A VHDL file will consist of a series of 'keywords' which are reserved and can't be used to identify a value or as a name of a signal, these are case sensitive so care is to be taken when entering. A comment line will start with two hyphens or in some cases two forward slashes and do not affect the compiler, an extra space or breaks in the lines also do not affect the compiler.

The declaration of the entity as briefly described above will designate the name of the entity along with identifying the Output and inputs, when entered into the system the entity will start with its key word which is 'entity', followed by the name given to the entity and ended with the word 'is' and shown below in figure 2.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Assn_1_6DM_tb is
4  -- Port ( ); No ned to add info here as
5  end Assn_1_6DM_tb;
6  architecture Behavioral of Assn_1_6DM_t
7  component Assn_1_6DM -- Enter file to b

```

Figure 2: Entity example

Following on from the entity declarations comes the port instructions, these again start with a key word which is 'port', as shown above in figure 2. The entity is then finished off ended with the key word as shown above in figure 2 'end', with the name of the entity after again shown above. Some words used in the entity are described as follows, the entity's name is up to the user or operator, an example is shown above in figure 2 were it has been named for the assignment in hand. Other words have a set meaning as follows, mode is used to represent the direction of the signal, 'in' is used as the input, 'out' is used for the output, 'buffer' is an output that can be detected and read in the architecture of the entity and 'input' is used as an input and/or a output. (Spiegel, 2001)

Other key words used to define the entity code are as follows;

Time = a time indication

Real = values of real origin

Character = a character print

Boolean = a false or true value

Integer = integer inputs

Bit = value of 0 or 1

Bit\_vector = vector from bit values

Std\_logic = nine values indicating the signal strength and value.

The architecture of the system is where the operations such as the operating variables and how they are implemented are defined, they can come in a structural or behavioural format. Below in figure 3 shows the architecture of a file used in the assignment and a behaviour model.

```
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 Asen_1_6 is
6  Port ( CLK100MHZ : in STD_LOGIC;
7        Ret : in std_logic;
8        FiveKHz : out STD_LOGIC
9  );
10 end Asen_1_6;
11 architecture Behavioral of Asen_1_6 is
12   signal cnt1 : integer := 0;
13   signal temp1 : std_logic := '0';
14   begin
15   process(CLK100MHZ,Ret)
16   begin
17     if Ret = '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 3: Architecture of system

The main heading of the model states the name of the architecture and also ties in the entity, as shown above in figure 3. The architecture then begins with the word begin giving a boolean expression, the text in the architecture uses logic expressions such as 'and or nand' and. The architecture will finish with the key word end followed by its model which as shown in figure 3 is a behavioural type. All VHDL files must end with a semi colon (;). (Spiegel, 2001)

A VHDL file can also include additional building blocks, these include a package part, a configuration part and a library part or section.

The package section is designed to gather all the parts of the file which can be shared globally with multiple other designs. The package will contain 2 design units, a body of the package and a declaration of the package. (Pak, 2000). (Pak, 2000) states that "A package declaration declares all the names of items that will be seen by the design units that use the package. A package body contains the implementation details of the subprograms declared in the package declaration".

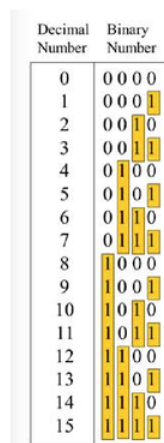
A configuration part to the VHDL can give the user the ability to modify variations of the package and select between different architectures. These configurations give quick substitutions of the design. (Pak, 2000)

*Very detailed research carried out*

## 2.2 Binary to BCD conversion.

Binary systems or code is a basic form of programming data used as coding for computer systems. The binary code is represented with either a one or a zero (1,0) called bits. Coding from machines and coding from binary are often brought together and create a raw code, this raw code is then used to work with hardware systems or computers. A binary bit is the simplest form of data and also carries or is the lowest value of data in a system, the ones and zeros combine to create strings which is then read by the hardware as coding. (Techopedia, 2018).

An eight-bit binary system or string has the ability to represent a value of up to 256. The binary system is a base 2 system and the bit on the far right is the least significant bit with the bit on the far left being the most significant bit. Below in figure 4 shows a 4-bit system and the decimal value for the binary representation up to the value of 15. (Bluegrass , 2017)



| Decimal Number | Binary Number |
|----------------|---------------|
| 0              | 0 0 0 0       |
| 1              | 0 0 0 1       |
| 2              | 0 0 1 0       |
| 3              | 0 0 1 1       |
| 4              | 0 1 0 0       |
| 5              | 0 1 0 1       |
| 6              | 0 1 1 0       |
| 7              | 0 1 1 1       |
| 8              | 1 0 0 0       |
| 9              | 1 0 0 1       |
| 10             | 1 0 1 0       |
| 11             | 1 0 1 1       |
| 12             | 1 1 0 0       |
| 13             | 1 1 0 1       |
| 14             | 1 1 1 0       |
| 15             | 1 1 1 1       |

Figure 4: 4 Bit Number to Binary Values

BCD or binary coded decimal is a type of code which is used to convert with number systems very accurately in a decimal form to be understood and read by the user more efficiently. A BCD system will create a four-digit string of binary code to represent one digit from 0-9 for unit's tens and hundreds. For example, a conversion of 15 as an 8-bit binary would be = 00001111 but as BCD it would be = 0001 0101.

(Rouse, 2017)

Binary to BCD conversion is needed as stated above to give a simpler usability or readability with the operator, with it giving this accurate representation it also has some disadvantages. The system becomes a lot more complex when adding this binary to BCD convertor as well as loosing storage space, BCD systems are getting used less often in modern time but as shown in the assignment the convertor has been used to display and easy readable representation of the LED's when using the switched as a binary input.

The Binary to BCD convertor used in the assignment with a VHDL system will take a binary input then from an interface and producer or convert into a BCD number of the same value as its output. Using the method of a base 2 system the binary bits which are inputted will shift one at a time with the base 2 system of doubling each time until the convertor has received it binary input then calculated the BCD equivalent for this number. The BCD digit can only have a maximum of the value 9 for each 4-bit indication, this means if we have a value higher than four, the base 2 system cannot be used so a value of 1 is added into the next BCD digit and then changed to suit the correct number or value. Figure 5 below shows a binary to BCD table. (Larson, 2017)

| Current Value |               | Next Value |               |
|---------------|---------------|------------|---------------|
| Decimal       | Register Bits | Decimal    | Register Bits |
| 0             | 0000          | 0          | 0 000X        |
| 1             | 0001          | 2          | 0 001X        |
| 2             | 0010          | 4          | 0 010X        |
| 3             | 0011          | 6          | 0 011X        |
| 4             | 0100          | 8          | 0 100X        |
| 5             | 0101          | 10         | 1 000X        |
| 6             | 0110          | 12         | 1 001X        |
| 7             | 0111          | 14         | 1 010X        |
| 8             | 1000          | 16         | 1 011X        |
| 9             | 1001          | 18         | 1 100X        |

Figure 5: Truth Table Binary - BCD (Larson, 2017)

*Again very detailed research carried out*

## 2.3 IP Block Design in the Vivado Design Suite.

Due to the increasing complexity that programs are now facing it is vital that a new method for designing and IP is used. The Vivado design IP block design system gives the operator or builder the ability to create complicated and advanced programs by connecting and instantiating the ip's from the advance design suite. These designs can then be created programmatically or interactive and are normally built to an interface standard to give better usability of the programs. The interface level is a combination of signals using the same functions.

This combination gives an advantage as if each signal was individual and not combined the system would become increasingly complex and the graphical indication would also be very complex. The combination of these signals and busses then makes it possible for one IP integrator to make a connection along with a slave and master connection and also a design rule check can be made to ensure the interface is running properly and connected properly. (Xilinx, 2013)

The IP block design also gives the ability to combine a number of different programs or VHDL files to work together and produce and output, for example to assignment requires a number of different VHDL files to be creates then packaged to an IP, these files were then saved as IP and opened in the block design mode were the design was built and the blocks interconnected which were then programmed to the nexys4 boards by generating bit streams and then opening the hardware manager to auto connect to the boards to produce the assignment requirements. Figure 6 below shows a block diagram used in the assignment for a binary to BCD 8-bit convertor.

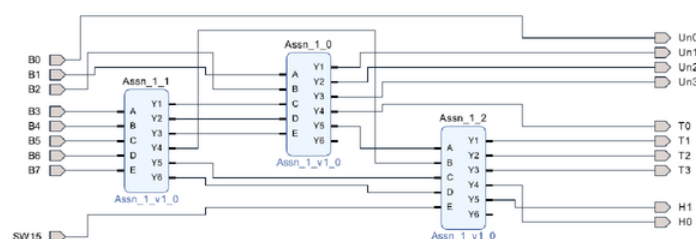


Figure 6: IP Block Design Binary - BCD

*Excellent detail*

### 3. Design of assignment requirements

Task 2 of the assignment requires a VHDL file to be created and simulated using a 74185A 6-bit binary to BCD convertor. The file was created using the techniques and skills learnt from the lecturers prior to the assignment on building Vivado systems.

The project was initially created by opening the program and selecting the 'Create Program' section from the pop up window that appears on opening the program. Once this function is selected the program then displays a series of pop-up windows asking for a location for the files to be saved to, this is important as the file name will need to match the name entered into the program code in order for the system to operate properly.

For the case of this task and assignment the name chosen was 'Assn\_1'. With the program named it is then selected to be operated as an 'RTL project'. The next step is to then select the board that the program is to connect with which in the case of the Nexys 4 DDR board was the 'xc7a100tcsg324-1' option. Following this the 'Finish' button is selected which opens the program to the selected variables, the target language is then changed from 'Verilog' to 'VHDL' and the program is ready to be built.

The general procedure for creating, simulating and testing is in the following methods;

- Create design
- Create simulation test bench
- Simulate the design
- Create a constraints file
- Synthesise the system
- Implement the system
- Bit stream to be generated
- Open the hardware manager
- Connect fpga and program
- Test the system on the board and record results

*Good detail*

For task 2 the assignment requires a 6-bit binary to BCD convertor to be created. This system will use 8 switched on the FPGA board and binary inputs and ten LED's as BCD outputs for the range of 0-255 with the far right LED being the least significant and the far left being the most significant. LED 0-3 will be the units, LED 4-7 will be the tens and LED 8 and 9 will be the hundreds.

The design of the VHDL system for the 6 bit was built for the assignment requirements and is shown below in figure 7, the green light in the right top corner shows the system works ok. The code will be explained as to what is being set and built and listed after the screenshot in figure 7.

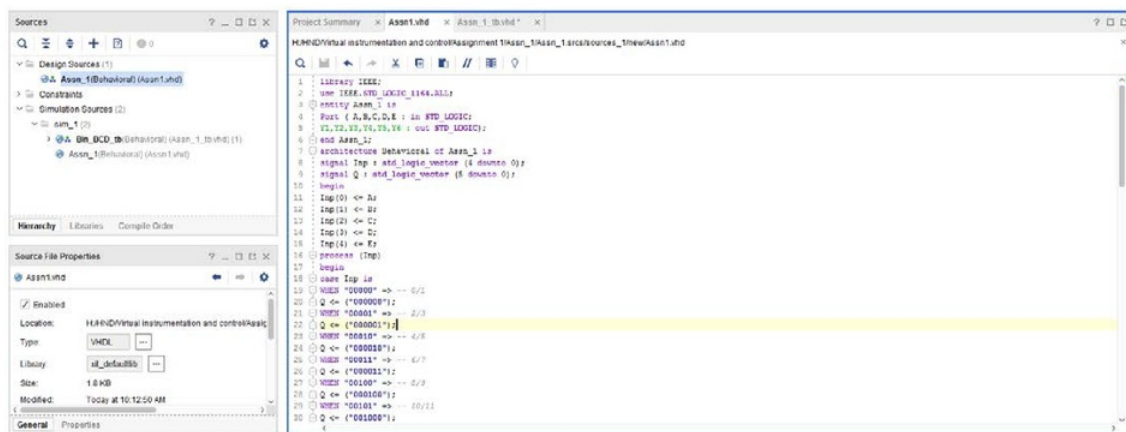


Figure 7: 6bit Binary to bcd design file

The above figure 7, shown a screenshot of the Vivado file created in the design section as shown in the left column under design sources and is set as a behavioural type, shown below figure 8.



Figure 8: Design Source

The first part of the design as shown below in figure 9 are as follows, the library is set to 'IEEE' this library stores all the information about the project such as its procedures, signals and operations. The package 'STD\_LOGIC\_1164' is defined in the library for 'std\_logic' so to use this file the file has to begin with the stating the library as 'IEEE' then telling it to 'use' 'STD\_LOGIC\_1164' as defined in the library, followed by 'ALL' telling to use every bit of information about this operation the library has. The code in the assignment is shown below in figure 9. (Spiegel, 2001)

*Good detail*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Assn_1 is
```

Figure 9: Library Commands

Following on from this comes the entity declaration, this is the stage where the name of the system is given and the outputs and inputs are listed. Figure 10 below shows the entity section.

```
entity Assn_1 is
    Port ( A,B,C,D,E : in STD_LOGIC;
          Y1,Y2,Y3,Y4,Y5,Y6 : out STD_LOGIC);
end Assn_1;
```

Figure 10: Entity 6bit

The entity has been named as 'Assn\_1' as shown above. The data has to start with the key word 'entity' and after the name finished with the key word 'is'. The port functions come next and they state the inputs which are, 'A,B,C,D,E' in standard logic for the binary input, then comes the outputs which are, 'Y1,Y2,Y3,Y4,Y5,Y6' for the BCD outputs. The entity is finished with the key word end followed by the name of the file as shown above in figure 10 and always finishes with a semi colon.

The architecture of the code comes next and is shown below in figure 11 and 12 of two screenshots of the start and end of the architecture.

```
architecture Behavioral of Assn_1 is
    signal Inp : std_logic_vector (4 downto 0);
    signal Q : std_logic_vector (5 downto 0);
begin
    Inp(0) <= A;
    Inp(1) <= B;
    Inp(2) <= C;
    Inp(3) <= D;
    Inp(4) <= E;
    process (Inp)
    begin
        case Inp is
            WHEN "00000" => -- 0/1
                Q <= ("000000");
            WHEN "00001" => -- 2/3
                Q <= ("000001");
            WHEN "00010" => -- 4/5
                Q <= ("000010");
            WHEN "00011" => -- 6/7
                Q <= ("000011");
            WHEN "00100" => -- 8/9
                Q <= ("000100");
            WHEN "00101" => -- 10/11
                Q <= ("001000");
        end case;
    end process;
```

Figure 11: 6 Bit Architecture start

```
            WHEN "11000" => -- 48/49
                Q <= ("100100");
            WHEN "11001" => -- 50/51
                Q <= ("101000");
            WHEN "11010" => -- 52/53
                Q <= ("101001");
            WHEN "11011" => -- 54/55
                Q <= ("101010");
            WHEN "11100" => -- 56/57
                Q <= ("101011");
            WHEN "11101" => -- 58/59
                Q <= ("101100");
            WHEN "11110" => -- 60/61
                Q <= ("110000");
            WHEN "11111" => -- 62/63
                Q <= ("110001");
            WHEN Others =>
                Q <= ("000000");
        end case;
    end process;

    Y1 <= Q(0);
    Y2 <= Q(1);
    Y3 <= Q(2);
    Y4 <= Q(3);
    Y5 <= Q(4);
    Y6 <= Q(5);
end behavioral;
```

Figure 12: 6 Bit Architecture End

The architecture is started with the key word 'Architecture' then sets it to a 'Behavioural' system of the entity name which was 'Assn\_1'. This is followed by the 'is' key word which states what declarations to be used and in this case, was two signal declarations of input 'std\_logic\_vector' 4 down to 0 and the same for 'Q' or output 5 down to 0.

The file is then set to start with the key word 'begin' and the inputs for binary are defined as above in figure 11. The <= symbol means the value on the right is equal or set to the value on the left and vice versa. The 'Process' key word is the used

which allows the program to use sequential statements as a description for the system.

The next part comes the 'case' statements for the system which is the conversion process, as seen in figure 11 and 12 the binary figure is set after the 'when' statement to followed by a comment to show its value and underneath the output of 'Q' showing the equivalent BCD value. This is continued for the decimal values up to 62/63 and then a final line stating that any other input will set the output to all 0's. an example is for the decimal value of 60/61 the binary input is '11110' and the BCD output is '110000', the system is finished with the statements 'End case' and 'End Process' for the convertor. The last part shown in figure 11 is showing that the BCD outputs which are Y1-Y6 are set to display when the 'Q' from the above code is activated for the correct number. To end the whole system the command 'end behavioural' is set. *Good detail*

The test bench file is next to be created, this is a simulation file so that the Vivado program can simulate the operation of the design before it is tested to ensure the operation is successful. The test bench file is created using the add sources and create simulation functions in the Vivado program setup, when naming the file, it has to be the same as the design file but with an underscore tb (\_tb) after it to indicate and run it as a simulation file. The file is then created and the pop up boxes are finished to display the file, all the original text in the test bench file is deleted and the new code is entered. The code below in figure 12 and 13 was entered and will be broken down and explained below. The code shown is from the word document and the entity name was changed to 'Assn\_1\_tb'.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Bin_BCD_tb is
    Port (
        -- Port 1;
        end Bin_BCD_tb;
    architecture Behavioral of Bin_BCD_tb is
        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT UUT_Bin_BCD
            Port ( A,B,C,D,E : in std_logic;
                  Y1,Y2,Y3,Y4,Y5,Y6 : out std_logic
                );
        END COMPONENT;

        --Inputs
        signal A : std_logic := '0';
        signal B : std_logic := '0';
        signal C : std_logic := '0';
        signal D : std_logic := '0';
        signal E : std_logic := '0';

        --Outputs
        signal Y1 : std_logic;
        signal Y2 : std_logic;
        signal Y3 : std_logic;
        signal Y4 : std_logic;
        signal Y5 : std_logic;
        signal Y6 : std_logic;

        --Clocks
        constant A_period : time := 1 us;
        constant B_period : time := 2 us;
        constant C_period : time := 4 us;
        constant D_period : time := 8 us;
        constant E_period : time := 16 us;

    BEGIN
        -- Instantiate the Unit Under Test (UUT)
        uut: Bin_BCD PORT MAP (
            A => A,
            B => B,
```

Figure 13: 6bit to BCD TB 1

```

            C => C,
            D => D,
            E => E,
            Y1 => Y1,
            Y2 => Y2,
            Y3 => Y3,
            Y4 => Y4,
            Y5 => Y5,
            Y6 => Y6
        );

        --Clock process definitions
        A_process: process
        begin
            A <= '0';
            wait for A_period/2;
            A <= '1';
            wait for A_period/2;
        end process;
        B_process: process
        begin
            B <= '0';
            wait for B_period/2;
            B <= '1';
            wait for B_period/2;
        end process;
        C_process: process
        begin
            C <= '0';
            wait for C_period/2;
            C <= '1';
            wait for C_period/2;
        end process;
        D_process: process
        begin
            D <= '0';
            wait for D_period/2;
            D <= '1';
            wait for D_period/2;
        end process;
        E_process: process
        begin
            E <= '0';
            wait for E_period/2;
            E <= '1';
            wait for E_period/2;
        end process;
    END;
```

Figure 14: 6 Bit to BCD TB 2

*Not very clear!*

Figure 15 below shows the first section of the code is the same as the VHDL file, the library declares to use the 'IEEE STD\_LOGIC\_1164' and the entity is the same but with the '\_tb' added behind the entity's name. the architecture then moves on to state the key word for the component under test which is 'component' followed by the 'port' statement and the binary inputs and BCD outputs which will be under test or simulated. The 'End component' statement is added which ends the declaration for what is going to be tested.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Bin_BCD_tb is
  -- Port ( );
end Bin_BCD_tb;

architecture Behavioral of Bin_BCD_tb is

  -- Component Declaration for the Unit Under Test (UUT)

  COMPONENT Bin_BCD
  Port ( A,B,C,D,E          : in std_logic;
        Y1,Y2,Y3,Y4,Y5,Y6 : out std_logic
        );
  END COMPONENT;
```

Figure 15: Test Bench entity

The program then goes on to list the binary inputs as shown below in figure 16 with a value given of '0', the outputs are then listed for the BCD values for Y1-Y6 for the 6 bits. A clock pulse is also entered to give a time period when simulating the model, the clock times are listed below in figure 16 along with the constant from A-E which will be used later on in the test bench to run the program when the program see's the period.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Bin_BCD_tb is
  -- Port ( );
end Bin_BCD_tb;

architecture Behavioral of Bin_BCD_tb is

  -- Component Declaration for the Unit Under Test (UUT)

  COMPONENT Bin_BCD
  Port ( A,B,C,D,E          : in std_logic;
        Y1,Y2,Y3,Y4,Y5,Y6 : out std_logic
        );
  END COMPONENT;
```

Figure 16: Inputs, Outputs and Clock 6bit

The program is then set to start with the key word 'Begin' the line starting with 'uut' is set which sets the unit under test for the simulation. The inputs for the binary and the outputs for the BCD are listed and defined then the clock pulse definitions are listed. The clock pulse beings by setting 'A' to '0' after then 'A\_period' which is displayed above, the system waits for this period then sets the rising edge at a '1' to output on the simulation. This is repeated for each input from A-E through all the clock periods as it looks for the inputs and what the corresponding outputs are. At the end of the simulation file it is finished with the command 'end process' and then 'END' to end the whole file.

```

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: Bto_BCD PORT MAP (
        A => A,
        B => B,
        C => C,
        D => D,
        E => E,
        Y1 => Y1,
        Y2 => Y2,
        Y3 => Y3,
        Y4 => Y4,
        Y5 => Y5,
        Y6 => Y6
    );

    -- Clock process definitions
    A_process: process
    begin
        A <= '0';
        wait for A_period/2;
        A <= '1';
        wait for A_period/2;
    end process;
    B_process: process
    begin
        B <= '0';
        wait for B_period/2;
        B <= '1';
        wait for B_period/2;
    end process;
    C_process: process
    begin
        C <= '0';
        wait for C_period/2;
        C <= '1';
        wait for C_period/2;
    end process;
    D_process: process
    begin
        D <= '0';
        wait for D_period/2;
        D <= '1';
        wait for D_period/2;
    end process;
    E_process: process
    begin
        E <= '0';
        wait for E_period/2;
        E <= '1';
        wait for E_period/2;
    end process;
END;

```

*Again, not very clear*

Figure 17: 6bit Unit Under Test

The simulation results for this 6-bit binary to BCD convertor are shown in part 4 of the assignment and the practical results are shown in part 5 of the assignment.

The next part of the assignment which was task 3, asked for the design to be adapted to become an 8-bit binary to BCD convertor. This was done by adapting the IP block design on Vivado. The addition of two extra binary to BCD convertors was needed in order to gain the 8 bits and output the correct BCD values. Below in figure 18 shows the new block design with the added Binary to BCD convertors.

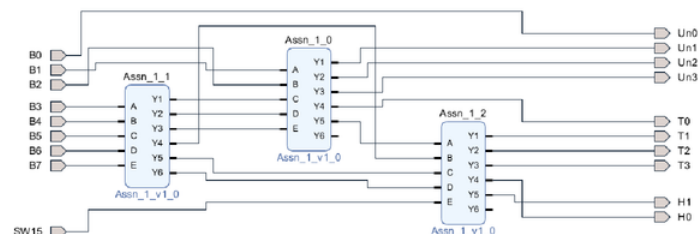


Figure 18: 8-bit binary to BCD convertor *Good design*

A Wrapper was created for the 8-bit counter by taking the block diagram and adding the additional binary to BCD convertors. Figure 19 below shows the new binary inputs which are B0-B7 which makes up 8 bits, switch 15 is then added which is the switch 15 on the Nexys 4 board. The outputs are then listed as 'out\_std\_logic' and represent the BCD values for units, tens and hundreds as shown on the code.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity Assn_1_3_wrapper is
    port (
        B0 : in STD_LOGIC;
        B1 : in STD_LOGIC;
        B2 : in STD_LOGIC;
        B3 : in STD_LOGIC;
        B4 : in STD_LOGIC;
        B5 : in STD_LOGIC;
        B6 : in STD_LOGIC;
        B7 : in STD_LOGIC;
        H0 : out STD_LOGIC;
        H1 : out STD_LOGIC;
        SW15 : in STD_LOGIC;
        T0 : out STD_LOGIC;
        T1 : out STD_LOGIC;
        T2 : out STD_LOGIC;
        T3 : out STD_LOGIC;
        Un0 : out STD_LOGIC;
        Un1 : out STD_LOGIC;
        Un2 : out STD_LOGIC;
        Un3 : out STD_LOGIC;
    );
end Assn_1_3_wrapper;
architecture STRUCTURE of Assn_1_3_wrapper is
    component Assn_1_3 is
```

Figure 19: 8bit Wrapper File

The second part of the code is then added which is for the 'architecture' and includes the values for the binary inputs and the BCD outputs being designated ports on the Nexys 4 board. This ties up with the constraints file which lists were what each port on the board is represented with on the vivado code files.

The constraints file came next, this is where the components on the Nexys 4 board are assigned values from the coding so that when downloaded to the FPGA the board will work correctly with the desired inputs and outputs operating the correct components on the board. Figure 20 below shows the constraints file with the switches and LED's corresponding to the code to ensure correct operation of the Nexys 4 board.

```
Switches

set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { B0 }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { B1 }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { B3 }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { B4 }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { B5 }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { B6 }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { B7 }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW15 }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Un0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { Un1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { Un2 }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { Un3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { T0 }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { T1 }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { T2 }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { T3 }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { H0 }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { H1 }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
```

Figure 20: 8bit Constraints

[Good detail shown on .xdc file design](#)

The test bench files were created for the 8-bit binary to BCD convertor and are shown below in figure 21, the program starts off with telling the system to use the library file for the 'STD\_LOGIC\_1164', the entity is then set to work for the same as the design file but with the tb added on as its saved as a test bench file. The binary inputs are listed after the 'port' key word to indicate the switched on the board and the outputs are also listed as 'out std\_logic' for the BCD outputs of units, tens and hundreds. For the test three values were chosen to be tested as per the assignment task and they are shown below in figure 21, The first block of testing is for the value of 232, this is shown in the binary form alongside the signals for B0 – B7.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Asn_1_3_tb is
-- Port ( );
end Asn_1_3_tb;
architecture Behavioral of Asn_1_3_tb is
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT Asn_1_3_wrapper
Port ( B0,B1,B2,B3,B4,B5,B6,B7,SW15 : in std_logic;
Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1 : out std_logic
);
END COMPONENT;

--Inputs
signal B0 : std_logic := '0';
signal B1 : std_logic := '0';
signal B2 : std_logic := '0';
signal B3 : std_logic := '1';
signal B4 : std_logic := '0';
signal B5 : std_logic := '1';
signal B6 : std_logic := '1';
signal B7 : std_logic := '1';
-- First test @ 232
signal SW15 : std_logic := '0';
--Outputs
signal Un0 : std_logic;
signal Un1 : std_logic;
signal Un2 : std_logic;
signal Un3 : std_logic;
signal T0 : std_logic;
signal T1 : std_logic;
<

```

Figure 21: 8bit Simulation, first test

The second and third test are shown below in figure 22 and test the values of 75 and 217. The code shows that the program is waiting for 1ms then the test at 75 is carried out, the binary values are then set at the value for 75 and 217 as indicated in binary 1's and 0's below. The process is then ended with the 'end process' command and the 'end behavioural' command.

```

);
-- Clock process definitions
-- No Clocks
-- Stimulus process
stim_proc: process
begin
wait for 1ms;
-- Second test @ 75
B0 <= '1';
B1 <= '1';
B2 <= '0';
B3 <= '1';
B4 <= '0';
B5 <= '0';
B6 <= '1';
B7 <= '0';
wait for 1ms;
-- Third test @ 217
B0 <= '1';
B1 <= '0';
B2 <= '0';
B3 <= '1';
B4 <= '1';
B5 <= '0';
B6 <= '1';
B7 <= '1';
wait;
end process;
end Behavioral;

```

*Testbench shown and clearly understood*

Figure 22: 8bit simulation, second and third test

#### 4. Analysis of Xilinx Vivado simulated results

For the 6-bit binary to BCD convertor the testing was carried out by synthesizing and generating a bit stream for the 6-bit binary to BCD convertor. For the first simulation, a value of 22/23 was picked to be tested. Below in figure 23 shows the simulation of moving the cursor along the generated table until the value of 22/23 is found as a binary input. The time scale was set to 16us.



Figure 23: 6bit 22/23 simulation

Figure 24 below then shows that the binary value for 22/23 as shown with A-E with A being the least significant and E being the most at 01011 1/0 and the BCD value as shown from Y1-Y6 with Y6 being the most significant and Y1 or Y0 being the least significant. The BCD value is 010001 1/0 as shown in the code design and as shown in binary to BCD conversion table.

| Name | Value |
|------|-------|
| A    | 1     |
| B    | 1     |
| C    | 0     |
| D    | 1     |
| E    | 0     |
| Y1   | 1     |
| Y2   | 0     |
| Y3   | 0     |
| Y4   | 0     |
| Y5   | 1     |
| Y6   | 0     |

Figure 24: 6bit 22/23 enhanced

For the second simulation, a value of 30/31 was picked to be tested. Below in figure 25 shows the simulation of moving the cursor along the generated table until the value of 30/31 is found as a binary input. The time scale was set to 16us.



Figure 25: 6 bit 30/31 Simulation

Figure 26 below then shows that the binary value for 30/31 as shown with A-E with A being the least significant and E being the most at 01111 1/0 and the BCD value as shown from Y1-Y6 with Y6 being the most significant and Y1 or Y0 being the least significant. The BCD value is 011000 1/0 as shown in the code design and as shown in binary to BCD conversion table.

| Name | Value |
|------|-------|
| A    | 1     |
| B    | 1     |
| C    | 1     |
| D    | 1     |
| E    | 0     |
| Y1   | 0     |
| Y2   | 0     |
| Y3   | 0     |
| Y4   | 1     |
| Y5   | 1     |
| Y6   | 0     |

Figure 26: 30/31 enhanced

For the Third simulation, a value of 8/9 was picked to be tested. Below in figure 27 shows the simulation of moving the cursor along the generated table until the value of 8/9 is found as a binary input. The time scale was set to 16us.

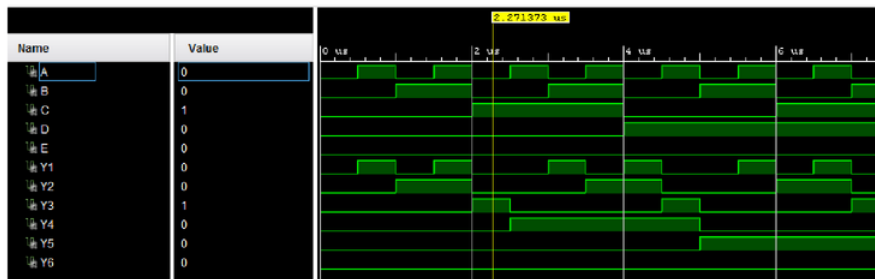


Figure 27: 6 bit 8/9 Sim

Figure 28 below then shows that the binary value for 8/9 as shown with A-E with A being the least significant and E being the most at 00100 1/0 and the BCD value as shown from Y1-Y6 with Y6 being the most significant and Y1 or Y0 being the least significant. The BCD value is 000100 1/0 as shown in the code design and as shown in binary to BCD conversion table.

| Name | Value |
|------|-------|
| A    | 0     |
| B    | 0     |
| C    | 1     |
| D    | 0     |
| E    | 0     |
| Y1   | 0     |
| Y2   | 0     |
| Y3   | 1     |
| Y4   | 0     |
| Y5   | 0     |
| Y6   | 0     |

Figure 28: 8/9 enhanced

*Three good set of results shown but are these results correct? Not enough detail in analysis*

The next stage of testing was for the 8-bit binary to BCD convertor, this required the same to be carried out with completing a synthesise and a generation of a bit stream after running the implantation. The testing was required for three values which were entered into the test bench file prior to running the bit stream as shown above in the assignment.

The first simulation test was for the value of 232, as it is the first test in the test bench file it is in the first-time frame column for 1us as the test bench file tests this at 1us as set in the program. The results are shown below in figure 29;

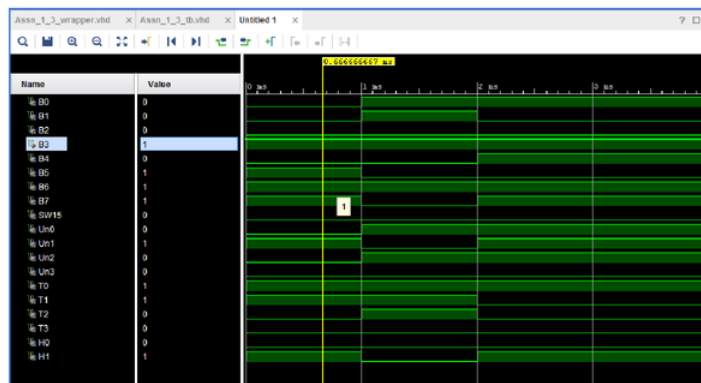


Figure 29: 8 bit 232 Sim

The binary value for 232 is '11101000', this is shown clearly below in figure 30 as correctly indicating this value. The BCD value is '1000110010' which is H1,H0,T3,T2,T1,T0,U3,U2,U1,U0 respectively, so we have a value of H1,T1,T0,U1, which is H1 to indicate 200, T0+T1 to give 1+2 (tens) to = 3 or 30, and U1, to indicate the value for 2 to give the number 2. So as shown above the table confirms this to give 232.

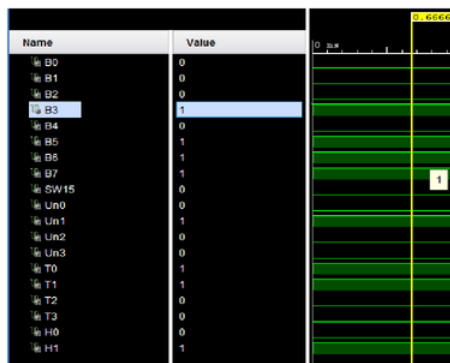


Figure 30: 232 Table

The second simulation test was for the value of 75, as it is the second test in the test bench file it is in the second-time frame column for 1us as the test bench file tests this at 1us as set in the program. The results are shown below in figure 31;

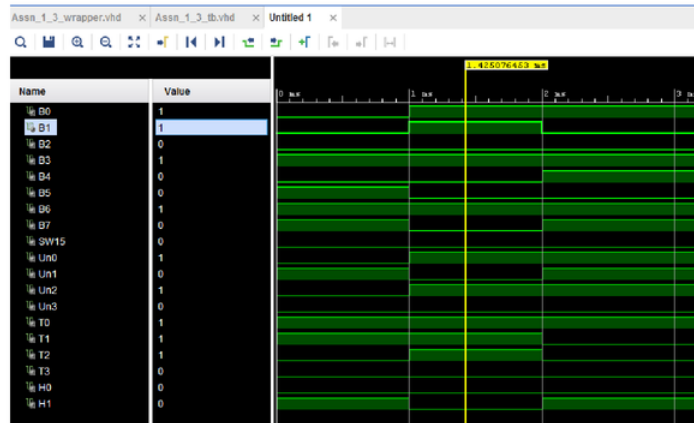


Figure 31: 8 bit, 75 Sim

The binary value for 75 is '01001011', this is shown clearly below in figure 32 as correctly indicating this value. The BCD value is '0001110101' which is H1,H0,T3,T2,T1,T0,U3,U2,U1,U0 respectively, so we have a value of  $T2+T1+T0+U2+U0$  which is,  $T2+T1+T0$  to give  $1+2+4$  (tens) to = 7 or 70, and  $U2+U0$ , to indicate the value for  $4+1$  to give the number 5. So as shown above the table confirms this to give 75.

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

Figure 32: 75 Sim enhanced

The third simulation test was for the value of 217, as it is the third test in the test bench file it is in the third-time frame column for 1us as the test bench file tests this at 1us as set in the program. The results are shown below in figure 33;

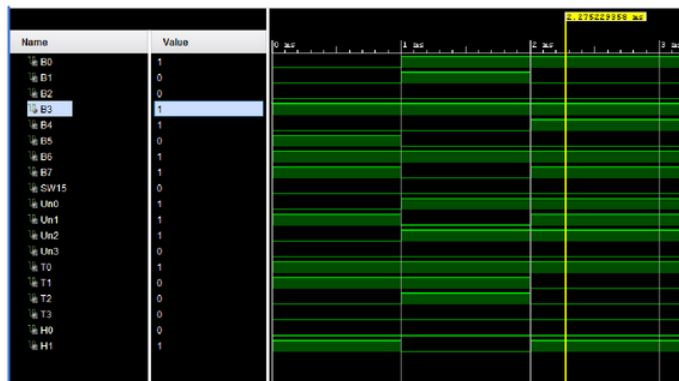


Figure 33: 8 bit, 217 Simulation

The binary value for 217 is '11011001', this is shown clearly below in figure 34 as correctly indicating this value. The BCD value is '1000010111' which is H1,H0,T3,T2,T1,T0,U3,U2,U1,U0 respectively, so we have a value of H1+T0+U2+U1+U0 which is, H1 to indicate 2 or 200 in the hundreds column, then T0 to give 1 or 10 in the tens column, and U2+U1+U0, to indicate the value for 4+2+1 to give the number 7. So as shown above the table confirms this to give 217.

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

Figure 34: 217 enhanced

*Excellent set of results which have been analysed well*

## 5. Analysis of Nexys4 FPGA practical results

Using the program designed and simulated above the next step to carry out the practical test is to connect the Vivado programs to the Nexys 4 boards. The way this is done is by selecting the 'open hardware manager' selection box on the Vivado program and programming the Nexys 4 board from the Vivado program. This requires a hard connection in the form of a cable from the USB port in the computer used with the loaded program running. This then communicated to the Nexys 4 board and programmed. The Nexys 4 board is shown below in figure 35 with a numbered diagram and key below.

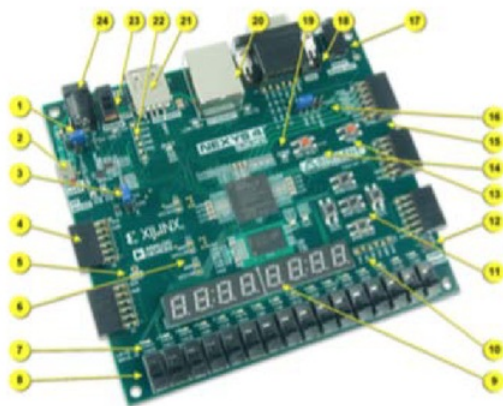


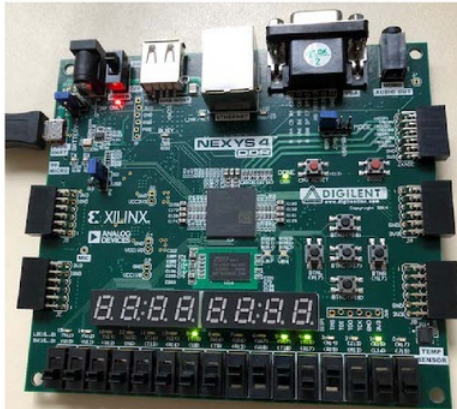
Figure 1. Nexys4 (DDR board) features.

| Callout | Component Description                    | Callout | Component Description                |
|---------|------------------------------------------|---------|--------------------------------------|
| 1       | Power select jumper and battery header   | 13      | FPGA configuration reset button      |
| 2       | Shared UART/JTAG USB port                | 14      | GPU reset button (for soft cores)    |
| 3       | External configuration jumper (SD / USB) | 15      | Analog signal Pmod port (KADC)       |
| 4       | Pmod port(s)                             | 16      | Programming mode jumper              |
| 5       | Microphone                               | 17      | Audio connector                      |
| 6       | Power supply test point(s)               | 18      | VGA connector                        |
| 7       | LEDs (16)                                | 19      | FPGA programming done LED            |
| 8       | Slide switches                           | 20      | Ethernet connector                   |
| 9       | Eight digit 7-seg display                | 21      | USB host connector                   |
| 10      | JTAG port for (optional) external cable  | 22      | PCI24 programming port (factory use) |
| 11      | Five pushbuttons                         | 23      | Power switch                         |
| 12      | Temperature sensor                       | 24      | Good detail                          |

Figure 35: Nexys 4 board Labelled

With the Nexys 4 board plugged into the computer and the design programmed to the board the practical testing could commence.

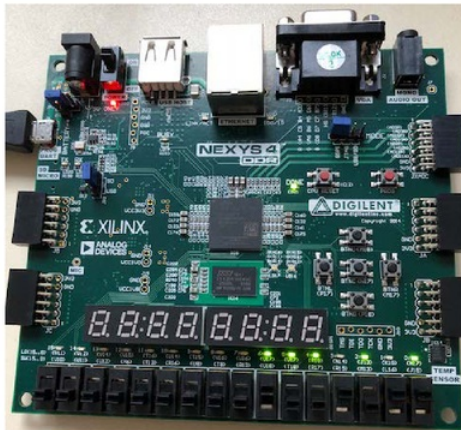
Below in figure 36 shows the first test for the value of 232 as simulated in the above part of the assignment. The binary inputs are as described before entered as the switches from SW0 – SW7 or as the simulated test shows, B0-B7. The binary input should be '11101000' on the switches which is shown and present below. The BCD output should be '1000110010' which from left to right on the board show a positive result and shown the binary value of 232.



Correct

Figure 36: 8-bit test 232

Below in figure 37 shows the first test for the value of 75 as simulated in the above part of the assignment. The binary inputs are as described before entered as the switches from SW0 – SW7 or as the simulated test shows, B0-B7. The binary input should be '01001011' on the switches which is shown and present below. The BCD output should be '0001110101' which from left to right on the board show a positive result and shown the binary value of 75.



Correct

Figure 37: 8-bit test 75

Below in figure 38 shows the first test for the value of 217 as simulated in the above part of the assignment. The binary inputs are as described before entered as the switches from SW0 – SW7 or as the simulated test shows, B0-B7. The binary input should be '11011001' on the switches which is shown and present below. The BCD output should be '100010111' which from left to right on the board show a positive result and shown the binary value of 217.

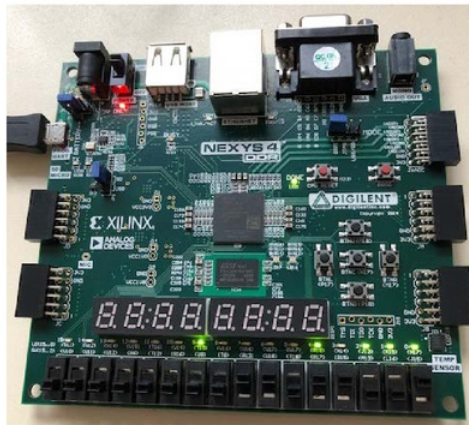


Figure 38: Test 8 bit 217

The testing provided apposite set of results and were as expected, this shows the Vivado programs built and the Simulations all worked correctly to produce this positive set of results for the simulations and practical testing.

*Very good set of practical results which  
have been well analysed*

## 6. Adapting design to suit new purposes or applications

The next part of the assignment was to adapt the original design built so that it could output the 8-bit binary range of 0 – 255 on a seven-segment display on the Nexys 4 board. This would require the addition of a 3-4-bit Mux\_Demux, a 5 KHz clock taken from the 100Mhz clock which is incorporated to the Nexys 4 board, and a BCD to seven segment displays. *Good*

The first stage in this adaption was the addition of the 5Khz clock, this uses the 100Mhz clock built in to the Nexys 4 board to become the 5Khz clock. The clock works with a count signal, this is an integer is a whole number.

*Incorrect*

The VHDL file below in figure 39 shows the 56Khz clock, the first part of the code is the inputs and outputs, as shown the 'Rst' is the input and the 'FiveKHz' as the output. The process sets the reset to '1' so if the systems sees the 1 it is reset back to the beginning. Each time period is measured in 10,000 counts after 10,000 counts it will change from a '1' to a '0' and vice versa, this is set for the 100Mhz. anything over 10000 and the system is reset. The frequency needs to be calculated to ensure the clock is running at 5khz and the code is correct, this is done with the calculation  $F=1/P$ , the calculation is to follow;

$$10,000 \times 1 / 100\text{Mhz}$$

$$= 10 \times 10^3 \times 10 \times 10^{-6}$$

$$= (100 \times 10^{-6}) \times 2 = P = 200 \times 10^{-6}$$

$$F = 1/P = 1/200 \times 10^{-6} = \frac{10^{-6}}{200} = 5\text{Khz} \quad \text{Very good!}$$

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;
entity Assn_1_6 is
Port ( CLK100MHZ : in STD_LOGIC;
Rst : in std_logic;
FiveKHz : out STD_LOGIC
);
end Assn_1_6;
architecture Behavioral of Assn_1_6 is
signal cnt1 : integer := 0;
signal temp1 : std_logic := '0';
begin
process (CLK100MHZ,Rst)
begin
if Rst = '1' then
temp1 <= '0';
cnt1 <= 0;
elsif rising_edge (CLK100MHZ) then
if(cnt1= 10000) then
temp1 <= not temp1;
cnt1 <= 0;
else
cnt1 <= cnt1 +1 ;
end if;
end if;
end process;
FiveKHz <= temp1;
end Behavioral;
```

Figure 39: 5khz clock

*Would have been nice to see a bit more detailed explanation on how the VHDL file functions*

The test bench file was now to be added which set out the test with the calculated frequency added. The inputs and output were added which were the same as the VHDL file. The UUT (unit under test) then gets added which is the 100Mhz clock and the five Khz clock, it then tests the clock while waiting for the period which is 200us for one time period to see the pulse on the simulation file to show operating at 5Khz

*Good detail*

```
use IEEE.std_logic_unsigned.all;
-- Document the following library declaration if instantiating
entity Asyn_1_6_tb is
  Port ( );
end Asyn_1_6_tb;
architecture Behavioral of Asyn_1_6_tb is
  Component Asyn_1_6
  Port ( CLK100MHZ : in STD_LOGIC;
        Rat : in STD_LOGIC;
        FiveKHz : out STD_LOGIC
        );
  end Component;
  --Inputs
  signal CLK100MHZ : STD_LOGIC := '0';
  signal Rat : STD_LOGIC := '1';
  --Outputs
  signal FiveKHz : STD_LOGIC;
  --Clock definitions
  constant CLK100MHZ_period : time := 10ns;
begin
  --Instantiate the UUT
  UUT : Asyn_1_6
  Port Map ( CLK100MHZ => CLK100MHZ,
            Rat => Rat,
            FiveKHz => FiveKHz
            );
  --Clock Process definitions
  CLK100MHZ_process : process
  begin
    CLK100MHZ <= '0';
    wait for CLK100MHZ_period/2;
    CLK100MHZ <= '1';
    wait for CLK100MHZ_period/2;
  end process;
  --Stimulus process definitions
  stim_proc : process
  begin
    wait for 1ms;
    Rat <= '0';
    wait for 2ms;
    Rat <= '1';
    wait;
  end process;
end Behavioral;
```

Figure 40: 5 kHz clock test bench

The simulation files will be added below in section 7 to shown the five 5Khz clock working from the 100Mhz clock for a time period.

The next part was the addition of the Mux – DeMux, this device allows the converted BCD units or numbers to be taken from the binary values and then sends them through to the seven segment display system as one input line. The multiplexer acts like a switchboard where it will take multiple inputs and connect them together internally then output them onto a single bus or line.

Mux and demux work differently in application, a mux can usually take a number of inputs and output them to one line and a demux can take one input and output that to multiple outputs.

Below is the VHDL file for the 3\*4-Bit Mux\_Demux.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;
entity Assn_1_6DM is
Port ( FiveKHz : in STD_LOGIC;
      Rat : in STD_LOGIC;
      AN0,AN1,AN2 : out STD_LOGIC;
      Un0,Un1,Un2,Un3 : in STD_LOGIC;
      T0,T1,T2,T3 : in STD_LOGIC;
      H0,H1 : in STD_LOGIC;
      A,B,C,D : out STD_LOGIC
);
end Assn_1_6DM;
architecture Behavioral of Assn_1_6DM is
    signal cnt : std_logic_vector (1 downto 0);
    signal Q : std_logic_vector (2 downto 0);
begin
    process (FiveKHz, Rat)
    begin
        if Rat = '1' then
            cnt <= "11";
        elsif rising_edge (FiveKHz) then
            cnt <= cnt + "01";
        end if;
    end process;
    process (cnt(1), cnt(0))
    begin
        if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
        elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
        elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
        else Q <= "111";
    end if;
    end process;
    AN0 <= Q(0);
    AN1 <= Q(1);
    AN2 <= Q(2);
    process (Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1,Q(0),Q(1),Q(2))
    begin
        if (Q(0) = '0' and Un0 = '0') then D<='0';
        elsif (Q(0) = '0' and Un0 = '1') then D<='1';
        elsif (Q(1) = '0' and T0 = '0') then D<='0';
        elsif (Q(1) = '0' and T0 = '1') then D<='1';
        elsif (Q(2) = '0' and H0 = '0') then D<='0';
        elsif (Q(2) = '0' and H0 = '1') then D<='1';
    end process;
end

```

Figure 41: Mux Demux VHDL

```

cnt <= cnt + "01";
end if;
end process;
process (cnt(1), cnt(0))
begin
if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
else Q <= "111";
end if;
end process;
A00 <= Q(0);
A01 <= Q(1);
A02 <= Q(2);
process (Un0,Un1,Un2,Un3,T0,T1,T2,T3,B0,H1,Q(0),Q(1),Q(2))
begin
if (Q(0) = '0' and Un0 = '0') then Dc<'0';
elsif (Q(0) = '0' and Un0 = '1') then Dc<'1';
elsif (Q(1) = '0' and T0 = '0') then Dc<'0';
elsif (Q(1) = '0' and T0 = '1') then Dc<'1';
elsif (Q(2) = '0' and B0 = '0') then Dc<'0';
elsif (Q(2) = '0' and B0 = '1') then Dc<'1';
end if;
if (Q(0) = '0' and Un1 = '0') then Cc<'0';
elsif (Q(0) = '0' and Un1 = '1') then Cc<'1';
elsif (Q(1) = '0' and T1 = '0') then Cc<'0';
elsif (Q(1) = '0' and T1 = '1') then Cc<'1';
elsif (Q(2) = '0' and H1 = '0') then Cc<'0';
elsif (Q(2) = '0' and H1 = '1') then Cc<'1';
end if;
if (Q(0) = '0' and Un2 = '0') then Bc<'0';
elsif (Q(0) = '0' and Un2 = '1') then Bc<'1';
elsif (Q(1) = '0' and T2 = '0') then Bc<'0';
elsif (Q(1) = '0' and T2 = '1') then Bc<'1';
elsif (Q(2) = '0') then Bc<'0';
end if;
if (Q(0) = '0' and Un3 = '0') then Ac<'0';
elsif (Q(0) = '0' and Un3 = '1') then Ac<'1';
elsif (Q(1) = '0' and T3 = '0') then Ac<'0';
elsif (Q(1) = '0' and T3 = '1') then Ac<'1';
elsif (Q(2) = '0') then Ac<'0';
end if;
end process;
end Behavioral;
end

```

Figure 42: MD VHDL 2

The code starts the same as the other code in that it defines the entity for the name and then lists the inputs and outputs. This mux\_Demux is taking 12 inputs as listed above in figure 41 and then has 7 outputs as also shown in figure 41.

The process is then beginning with the five5Khz clock as an 'elsif' statement looking for the rising edge. The reset is set to reset to 11 or carry on in steps of 1 in the else statement. The following part of the code takes the values for the hundreds, tens and units from the convertors and multiplexes them to sort into four lines of code which are send to the seven-segment display as the final output on the Nexys 4 board.

*More detail required!*

The test bench files below in figure 43 and 44 show the test bench with the number 193 to be used as a test number to be simulated on the vivado simulation file. The code starts with the inputs and outputs again being declared, the code then states the BCD value for 193 in terms of hundreds, tens and units. This is shown below in figure 43.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity Assn_1_6DM_tb is
4  -- Port ( ); No need to add info here as this is a testbench
5  end Assn_1_6DM_tb;
6  architecture Behavioral of Assn_1_6DM_tb is
7  component Assn_1_6DM -- 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 := '1';
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 := '0';
29  signal T3 : std_logic := '1';
30  signal H0 : std_logic := '1';
31  signal H1 : std_logic := '0';
32  -- 193
33  -- output signals
34  -- Example signal Q1 : std_logic := '0';
35  signal AN0 : std_logic;
36  signal AN1 : std_logic;
37  signal AN2 : std_logic;
38  signal A : std_logic;
39  signal B : std_logic;
40  signal C : std_logic;
41  signal D : std_logic;
42  -- clock period definitions
43  -- Example constant A_period : time := 8ns;
44  constant FiveKHz_period : time := 200us;
45  begin
46

```

Figure 43: MD simulation file

The below test bench file in figure 44 then shows were the BCD units are assigned to and the process is set to work over the 5Khz then its ended.

```

14 constant FiveKHz_period : time := 200us;
15 begin
16 -- Instantiate the UUT
17 UUT : Asn_1_6DM -- Enter file to be tested
18 Port Map (
19 -- Example A => A,
20 FiveKHz => FiveKHz,
21 Rst => Rst,
22 AN0 => AN0,
23 AN1 => AN1,
24 An2 => AN2,
25 Un0 => Un0,
26 Un1 => Un1,
27 Un2 => Un2,
28 Un3 => Un3,
29 T0 => T0,
30 T1 => T1,
31 T2 => T2,
32 T3 => T3,
33 H0 => H0,
34 H1 => H1,
35 A => A,
36 B => B,
37 C => C,
38 D => D
39 );
40 -- clock process definitions
41 clk_process : process
42 begin
43 FiveKHz <= '0';
44 wait for FiveKHz_period/2; --for 5 ns signal is '0'.
45 FiveKHz <= '1';
46 wait for FiveKHz_period/2; --for next 5 ns signal is '1'.
47 end process;
48 -- Stimulus process
49 stim_proc: process
50 begin
51 wait for 1ms;
52 Rst <= '0';
53 wait for 5ms;
54 Rst <= '1';
55 wait;
56 end process;
57 end Behavioral;

```

*Good detail*

Figure 44: DM test bench 2

The file is simulated as shown in part 7 of the assignment below.

The last part to be built is the BCD to 7 segment display. This part of the system will take its inputs from the mux\_demux and transmit them to the correct segment on the three seven segments that are to be used in the assignment. The VHDL design code is shown below in figure 45 and explained after.

```

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

```

Figure 45: 7 Segment display VHDL 1

The above VHDL code shown the inputs which are four inputs from the Mux\_Demux, and are from A-D, the outputs are the segments parts for the 7-segment display from sa-sg. The following piece of code is for the 'Q' values which are 0-1 and d-a respectively.

*Good detail*

```

20      Q(1) <= C;
21      Q(2) <= B;
22      Q(3) <= A;
23
24  process (Q)
25  begin
26      case Q is
27          WHEN "0000" =>          -- 0
28              seg <= ("0000001");
29          WHEN "0001" =>          -- 1
30              seg <= ("1001111");
31          WHEN "0010" =>          -- 2
32              seg <= ("0010010");
33          WHEN "0011" =>          -- 3
34              seg <= ("0000110");
35          WHEN "0100" =>          -- 4
36              seg <= ("1001100");
37          WHEN "0101" =>          -- 5
38              seg <= ("0100100");
39          WHEN "0110" =>          -- 6
40              seg <= ("0100000");
41          WHEN "0111" =>          -- 7
42              seg <= ("0001111");
43          WHEN "1000" =>          -- 8
44              seg <= ("0000000");
45          WHEN "1001" =>          -- 9
46              seg <= ("0001100");
47          WHEN Others =>          -- 'blank'
48              seg <= ("1111111");
49      end case;
50  end process;
51
52
53      sa <= seg(6);
54      sb <= seg(5);
55      sc <= seg(4);
56      sd <= seg(3);
57      se <= seg(2);
58      sf <= seg(1);
59      sg <= seg(0);
60
61  end Behavioral;
62
63

```

The above figure 46 shows the bottom section of the VHDL code being assigned binary numbers for the seven-segment value for each number, the top part after the 'When' statement is the binary number and the statement under with the key word 'seg' is the output to the segments with '0' making a segment active and '1' making a segment off. The bottom part of figure 46 shows each segment value from 'sa-sg' being assigned a segment number from '6-0'.

The simulation file is shown below in figure 47, shows the simulation file with it starting the same as other test bench files by declaring the entity name, then listing the inputs to be used for A-D and the outputs for the 7 segments for 'sa-sg'. The code then assigns the segments to the constraints file to be added below to operate the Nexys 4 board.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  --USE ieee.numeric_std.ALL;
7
8  ENTITY BCD_7seg_tb IS
9  END BCD_7seg_tb;
10
11 ARCHITECTURE behavior OF BCD_7seg_tb IS
12
13     -- Component Declaration for the Unit Under Test (UUT)
14
15     COMPONENT BCD_7seg
16     PORT (
17         A : IN  std_logic;
18         B : IN  std_logic;
19         C : IN  std_logic;
20         D : IN  std_logic;
21         sa : OUT std_logic;
22         sb : OUT std_logic;
23         sc : OUT std_logic;
24         sd : OUT std_logic;
25         se : OUT std_logic;
26         sf : OUT std_logic;
27         sg : OUT std_logic;
28     );
29     END COMPONENT;
30
31
32     --Inputs
33     signal A : std_logic := '0';
34     signal B : std_logic := '0';
35     signal C : std_logic := '0';
36     signal D : std_logic := '0';
37
38     --Outputs
39     signal sa : std_logic;
40     signal sb : std_logic;
41     signal sc : std_logic;
42     signal sd : std_logic;
43     signal se : std_logic;
44     signal sf : std_logic;
45     signal sg : std_logic;

```

Figure 46: 7 segment Simulation

```

62      sa => sa,
63      sb => sb,
64      sc => sc,
65      sd => sd,
66      se => se,
67      sf => sf,
68      sg => sg
69    );
70
71    -- Clock process definitions
72    A_process :process
73    begin
74      A <= '0';
75      wait for A_period/2;
76      A <= '1';
77      wait for A_period/2;
78    end process;
79
80    B_process :process
81    begin
82      B <= '0';
83      wait for B_period/2;
84      B <= '1';
85      wait for B_period/2;
86    end process;
87
88    C_process :process
89    begin
90      C <= '0';
91      wait for C_period/2;
92      C <= '1';
93      wait for C_period/2;
94    end process;
95
96    D_process :process
97    begin
98      D <= '0';
99      wait for D_period/2;
100     D <= '1';
101     wait for D_period/2;
102   end process;
103
104 end behavior;
105

```

Figure 47: 7 Seg Sim 2

Figure 47 above then shows the Unit under test operations which switches A-D and 0-1. The clock pulse is then introduced which is monitoring for any value change to output to the segment displays. This pulse and rising edge along with falling edge causes the change in value that the code will see.

*Not a good explanation!*

The constraints file is last, this is the file which tells the system what each statement is set to on the board, an example is what switch number or segment number a value on the code is assigned to. Below in figure 48 shows the constraints files used with the assigned values for the inputs and outputs.

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { D }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { C }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { A }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { SW0 }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { SW1 }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { SW2 }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { SW3 }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { SW4 }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { SW5 }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW6 }]; #IO_L15P_T2_DQS_RDMR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW7 }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]

##7 segment display

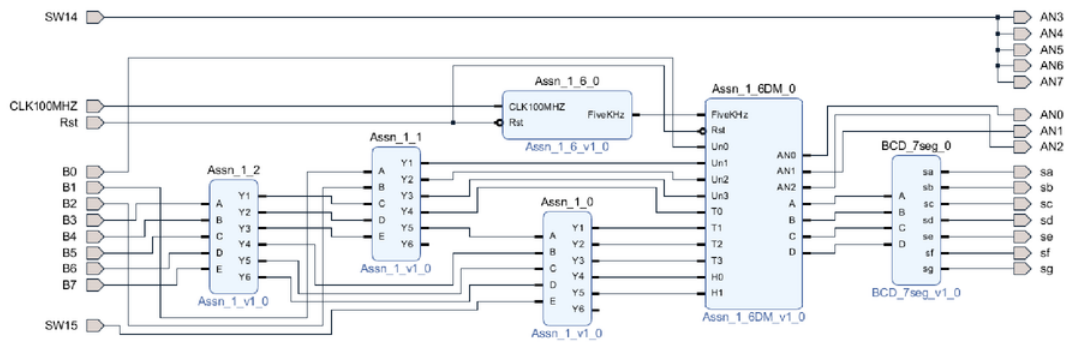
set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { sa }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { sb }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { sc }]; #IO_25_15 Sch=cc
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]
```

Figure 48: 7 Segment Constraints

*Very good detail*

With all the additional files built they were simulated in section 7 of this assignment and then tested to the Nexys 4 board to carry out the practical test. The in order to carry this test the files would all need to be brought together into one new program to test them as a whole unit. This meant saving all the files as I/P's as done earlier in the assignment then built and connected using the Vivado block design function. The program is shown below in figure 49.



Block diagram correct

Figure 49: I/P block design for adaptation for 7 seg

The above figure 49 shows the block diagram, as shown earlier in the assignment each part of the assignment was brought together and loaded into this block design package from saving as an I/P, they are found by searching the library for the name they were saved under. Once each part was loaded into the design they were connected up as shown above with binary inputs, the 5Khz clock inputs and switch 14 and 15. This I/P block design mode makes it a lot simpler to connect up the components and each VHDL and tb file is already pre-loaded into the designs. The design was then tested for the three original values used in task 4 and shown in the testing section 8 below.

## 7. Analysis of Xilinx Vivado simulated results

The first simulation was for the 5khz clock, as explained in the above section the clock is looking for 1 time period which has shown below was for a period of 200us.

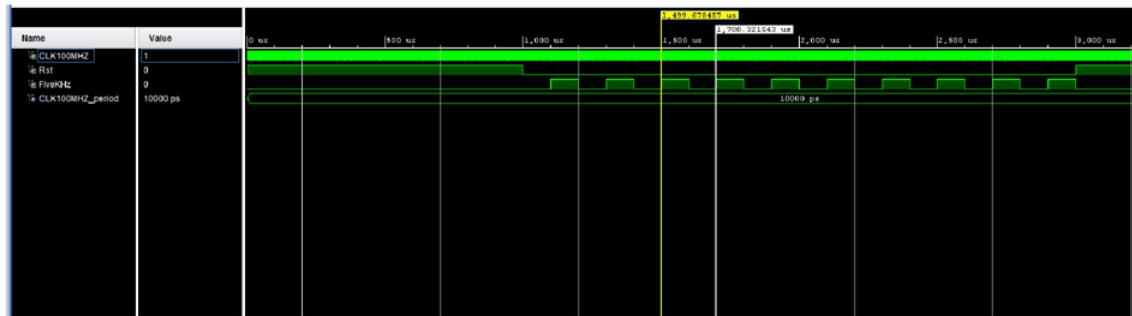


Figure 50: Simulation of 5khz clock

With the time found at 200us with the two cursors as shown above the formulae was used as above in the assignment to work out the frequency of the clock, the formulae were as follows;

$$\begin{aligned}\text{Frequency} &= 1/P \\ &= 1/200 \times 10^{-6} = 5000 \\ &= 5\text{Khz}\end{aligned}$$

*Good result*

This proves the files have worked and the clock is now set for a 5khz clock to use with the other parts of the program.

Next came the simulation for the Mux\_Demux files, this involved using the test bench files for the number selected to test which was 193, the bit stream was generated and the results were shown below in figure 51.

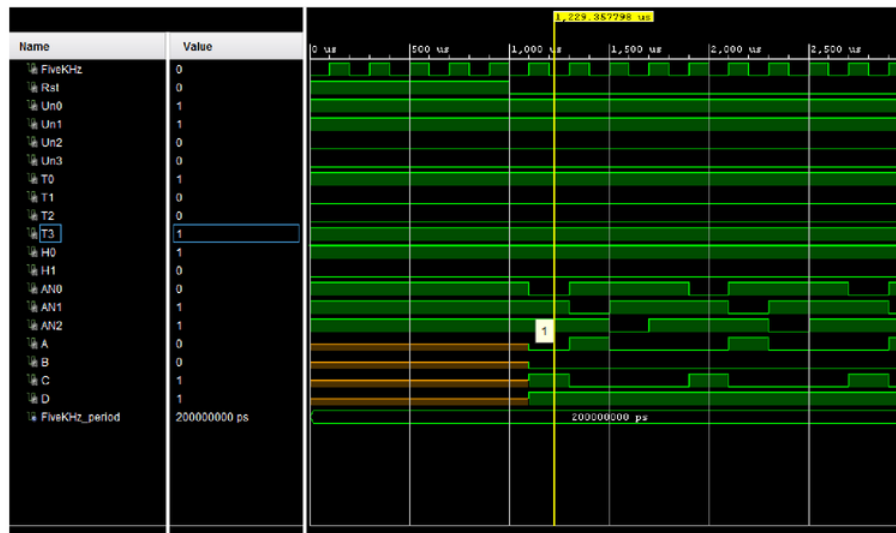


Figure 51: Mux\_demux Sim

Above shows the value for 193 displayed on the chart and represented by then unit's tens and hundreds column on the left-hand side. Then below shows the display showing D, C which are least significant at D and most at A, so D represent 1 and C represents 2, so the total value is displayed at 3 on the screen which represents the units.

*Good detail*

| Name    | Value |
|---------|-------|
| FiveKHz | 0     |
| Rst     | 0     |
| Un0     | 1     |
| Un1     | 1     |
| Un2     | 0     |
| Un3     | 0     |
| T0      | 1     |
| T1      | 0     |
| T2      | 0     |
| T3      | 1     |
| H0      | 1     |
| H1      | 0     |
| AN0     | 0     |
| AN1     | 1     |
| AN2     | 1     |
| A       | 0     |
| B       | 0     |
| C       | 1     |
| D       | 1     |

Figure 52: Value of 3

The second part was to simulate the file and move the cursor for the value of tens as shown below in figure 53;

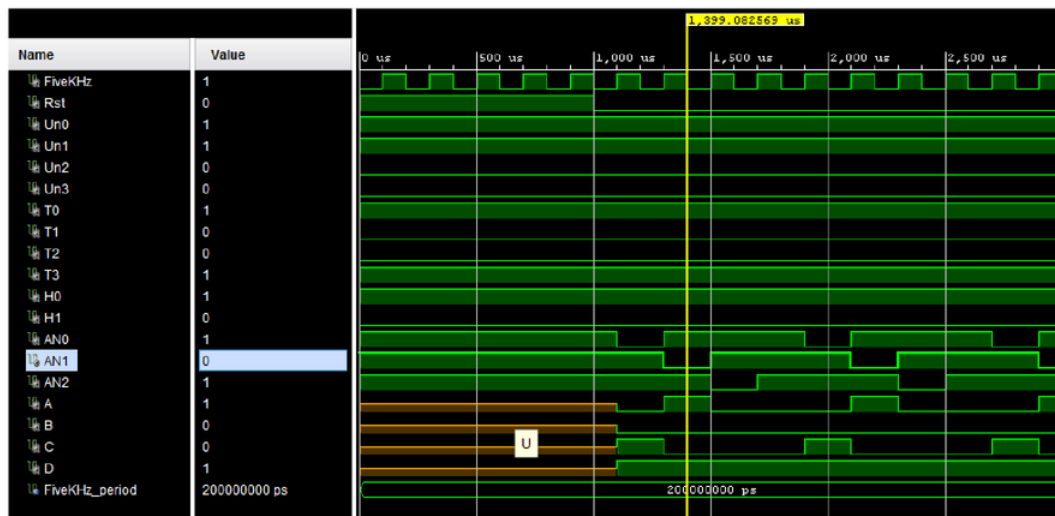


Figure 53: Mux\_demux tens

Above shows the value for 193 displayed on the chart and represented by then unit's tens and hundreds column on the left-hand side. Then below shows the display showing D, A, which are least significant at D and most at A, so D represent 1 and A represents 8, so the total value is displayed at 9 on the screen which represents the tens.

*Good understanding shown*

| Name           | Value        |
|----------------|--------------|
| FiveKHz        | 1            |
| Rst            | 0            |
| Un0            | 1            |
| Un1            | 1            |
| Un2            | 0            |
| Un3            | 0            |
| T0             | 1            |
| T1             | 0            |
| T2             | 0            |
| T3             | 1            |
| H0             | 1            |
| H1             | 0            |
| AN0            | 1            |
| AN1            | 0            |
| AN2            | 1            |
| A              | 1            |
| B              | 0            |
| C              | 0            |
| D              | 1            |
| FiveKHz_period | 200000000 ps |

Figure 54: Table value for tens

The third part was to simulate the file and move the cursor for the value of hundreds as shown below in figure 55;



Figure 55: Mux\_Demux hundreds

Above shows the value for 193 displayed on the chart and represented by then unit's tens and hundreds column on the left-hand side. Then below shows the display showing D, which are least significant at D and most at A, so D represent 1, so the total value is displayed at 1 on the screen which represents the hundreds

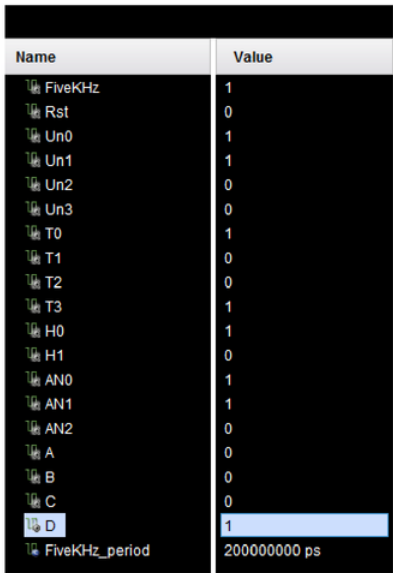


Figure 56: Hundreds mux\_demux

Again good understanding shown

The final simulation was for the seven-segment display, this took the test bench file with an A-B input from the mux\_demux and outputted to 7 segments on the display, the result is shown below in the screenshots. Three values were chosen to be displayed on the simulation.

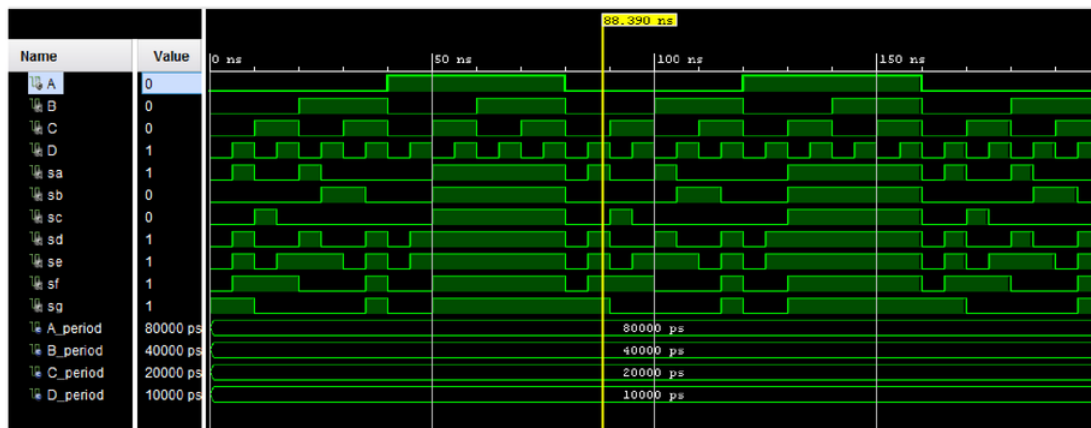


Figure 57: 7 Seg Sim value 1

Above in figure 57 shows the binary value for 1 which is represented as D, *Incorrect sa=1*  
The display shows sa=0, sb=0, sc=0, sd=1, se=1, sf=1, sg=1 with 1 not displaying and 0 displaying on the display. This proves that the seven segments would display the value of 1 if tested to the board.

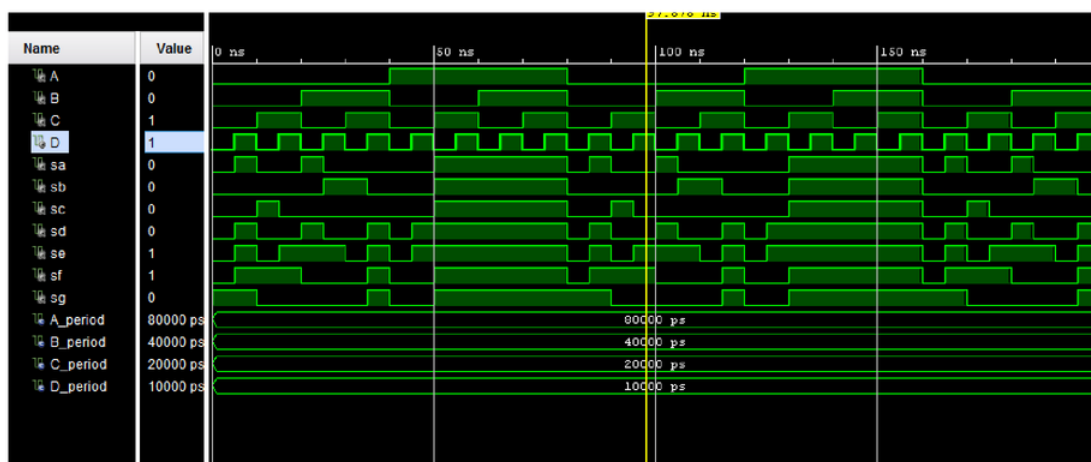


Figure 58: 7 seg value of 3

Above in figure 58 shows the binary value for 3.  
The display shows (sa =0, sb =0, sc =0, sd =0, se =1, sf =1, sg =0) with 1 not displaying and 0 displaying on the display. This proves that the seven segments would display the value of 3 if tested to the board.

*Good analysis*

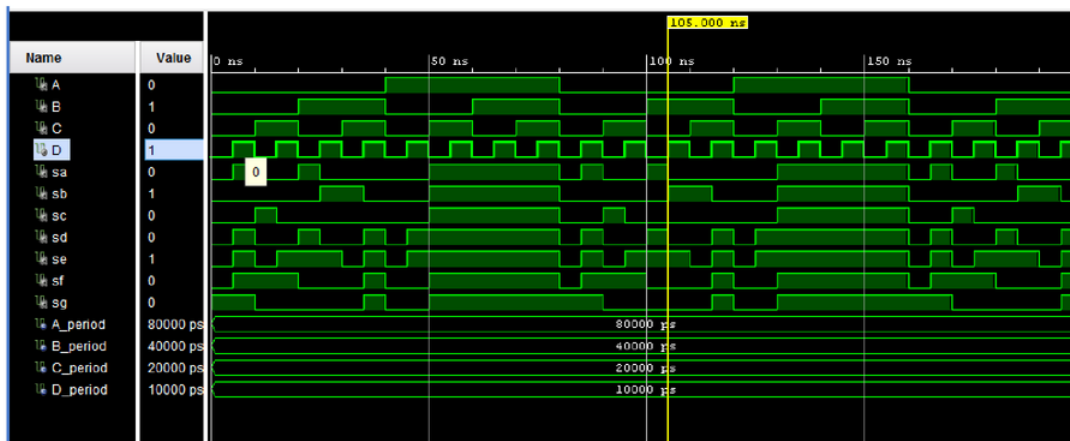


Figure 59: 7 Seg Value of 5

Above in figure 59 shows the binary value for 5.

The display shows (sa =0, sb =1, sc =0, sd =0, se =1, sf =0, sg =0) with 1 not displaying and 0 displaying on the display. This proves that the seven segments would display the value of 5 if tested to the board.

All the simulations were as expected and no difficulties were encountered, this means all the files are now ready to be practically tested to the Nexys 4 board as shown in the next part 8 of the assignment.

*Again good understanding shown*

## 8. Analysis of Nexys 4 DDR FPGA practical results

Following on to task 7 in the assignment the program created in the block diagram mode with all the component inserted was now to be downloaded to the Nexys 4 DDR FPGA board. The same three values were used as used in the original practical test for the binary to BCD convertor in part 5 of the assignment, the practical results are as follows.

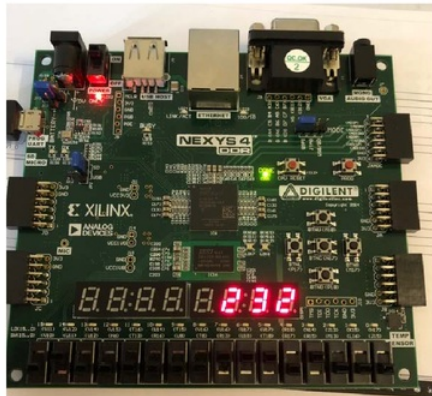


Figure 60: Practical 7 seg 232

*Correct result... why is SW14 set to logic 1?*

Figure 60 above shows the seven-segment display working and producing the value of 232, this was done with the binary switch inputs as below in figure 61 represents. The switches should be '11101000' which as shown above are correct.

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

Figure 61: 232 Value sim table

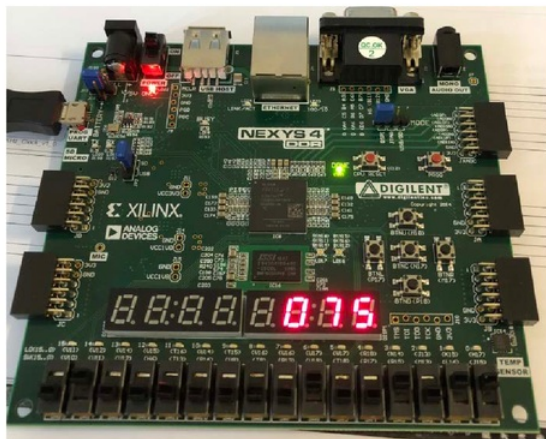


Figure 62: Simulation for 075

Good result

Figure 62 above shows the seven-segment display working and producing the value of 075, this was done with the binary switch inputs as below in figure 63 represents. The switches should be '01001011' which as shown above are correct.

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

Figure 63: Table for 075 Sim

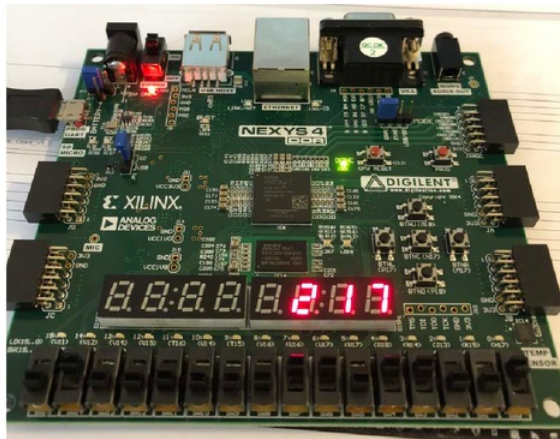


Figure 64: 7 seg practical for 217

*Good use made of pictorial evidence*

Figure 64 above shows the seven-segment display working and producing the value of 217, this was done with the binary switch inputs as below in figure 65 represents. The switches should be '11011001' which as shown above are correct.

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

Figure 65: Simulation table for the value of 217

The simulations and practical were the same and proved that the vivado programs built were built correctly and have met the assignment brief on what was expected to be found. The results were as expected and no problems were encountered.

*Good detail*

## 9. Conclusions

The assignment on a whole proved to be very helpful and a lot was learnt from carrying out the research and completing the design, simulations and testing. All results that were obtained were as expected and no difficulties or problems were found.

The research gave a better understanding of how Vivado works and the structure of a VHDL file, reading from sources other than that learn in the lecture and using the program provided a broader understanding and was helpful for the tasks that followed by providing more knowledge. This research was proved back by carrying out the designs and simulations by seeing how the program worked compared to the knowledge obtained, an example would be carrying out the construction of an 8bit and 6bit binary to BCD convertor on Vivado and seeing how the program takes a binary number and turns them into a BCD value. This was researched and proved back to compare and back up the research.

Carrying out the construction of the 6Bit binary to BCD convertor and then adapting that to become an 8-bit binary to BCD convertor gave me the skills and knowledge of how it is done and why it is done. The results that were simulated then proved the operation which meant the programs were correct and worked as expected, the practical results also world as expected and the real word use of the system was shown and understood.

By further adapting the programme to produce a 7-segment display was a valuable bit of knowledge as this showed a small change to a LED board takes a lot of programming and different components working together to achieve the desired outcome. The systems all worked as expected and the simulation of each file proved that they worked individually before they came together to give the final practical outcome on the Nexys 4 board.

Overall the assignment was a success, a lot was learnt from the research and the practical sessions carried out at college, with further adaption the program could have more uses and with the skills gained in this assignment different types of programmes could be built for different real-world situations. I feel I have explained the program to the best of my ability and with more practice and experience using the system the terminology and deployment could be improved.

*Good general conclusion which also contained some good reflection*

## 10. Appendices including bibliography

### Works Cited

Bluegrass , 2017. *Numbering Sytems*. [Online]

Available at:

[http://district.bluegrass.kctcs.edu/kevin.dunn/files/Numbering\\_Systems/2\\_Numbering\\_Systems\\_print.html](http://district.bluegrass.kctcs.edu/kevin.dunn/files/Numbering_Systems/2_Numbering_Systems_print.html)

[Accessed 16 01 2018].

Larson, S., 2017. *Binary to BCD Convertor (VHDL)*. [Online]

Available at: <https://eewiki.net/pages/viewpage.action?pageId=60030986>

[Accessed 15 01 2018].

Pak, B., 2000. *VHDL*. [Online]

Available at: <http://web.itu.edu.tr/~ateserd/VHDL.pdf>

[Accessed 14 01 2018].

Rouse, M., 2017. *Binary Coded Decimal*. [Online]

Available at: <http://whatis.techtarget.com/definition/binary-coded-decimal>

[Accessed 15 01 2018].

Spiegel, J. V. d., 2001. *VHDL Tutorial*. [Online]

Available at: [https://www.seas.upenn.edu/~ese171/vhdl/vhdl\\_primer.html](https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html)

[Accessed 14 01 2018].

Techopedia, 2018. *Binary code*. [Online]

Available at: <https://www.techopedia.com/definition/17052/binary-code>

[Accessed 16 01 2018].

Xilinx, 2013. *Vivado deisgn Suite user guide*. [Online]

Available at:

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx2013\\_3/ug994-vivado-ip-subsystems.pdf](https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug994-vivado-ip-subsystems.pdf)

[Accessed 15 01 2018].

*Good list of citations which shows research carried out*

FINAL GRADE

GENERAL COMMENTS

Instructor

65/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

**Text Comment.** to be

**Text Comment.** Good, clear introduction given

PAGE 7

PAGE 8

PAGE 9

PAGE 10

**Text Comment.** Very detailed research carried out

PAGE 11

PAGE 12

**Text Comment.** Again very detailed research carried out

PAGE 13

**Text Comment.** Excellent detail

PAGE 14

**Text Comment.** Good detail

PAGE 15

---

**Text Comment.** Good detail

PAGE 16

---

PAGE 17

---

**Text Comment.** Good detail

**Text Comment.** Not very clear!

PAGE 18

---

PAGE 19

---

**Text Comment.** Again, not very clear

PAGE 20

---

**Text Comment.** Good design

PAGE 21

---

**Text Comment.** Good detail shown on .xdc file design

PAGE 22

---

**Text Comment.** Testbench shown and clearly understood

PAGE 23

---

PAGE 24

---

PAGE 25

---

**Text Comment.** Three good set of results shown but are these results correct? Not enough detail in analysis

PAGE 26

---

PAGE 27

---

PAGE 28

---

**Text Comment.** Excellent set of results which have been analysed well

**Text Comment.** Good detail

**Text Comment.** Correct

**Text Comment.** Correct

**Text Comment.** Very good set of practical results which have been well analysed

**Text Comment.** Good

**Text Comment.** Incorrect

**Text Comment.** Very good!

**Text Comment.** Would have been nice to see a bit more detailed explanation on how the VHDL file functions

**Text Comment.** Good detail

**Text Comment.** More detail required!

**Text Comment.** Good detail

**Text Comment.** Good detail

**Text Comment.** Not a good explanation!

**Text Comment.** Very good detail

**Text Comment.** Block diagram correct

**Text Comment.** Good result

**Text Comment.** Good detail

**Text Comment.** Good understanding shown

**Text Comment.** Again good understanding shown

**Text Comment.** Incorrect  $sa=1$

**Text Comment.** Good analysis

**Text Comment.** Again good understanding shown

**Text Comment.** Correct result... why is SW14 set to logic 1?

**Text Comment.** Good result

**Text Comment.** Good use made of pictorial evidence

**Text Comment.** Good detail

PAGE 53

**Text Comment.** Good general conclusion which also contained some good reflection

PAGE 54

**Text Comment.** Good list of citations which shows research carried out

GRADING FORM: CSG-RUBIC

**NATHAN SMITH**

**65**

#### RESEARCH-INFORMED LITERATURE



Structure of a typical VHDL file: Very good, detailed research carried out. (3/5) Binary to BCD conversion: Again very detailed research carried out. (3/5) IP Block Design in Vivado: Excellent detail shown in research. (4/5) Total (10/15)

**10**

#### KNOWLEDGE AND UNDERSTANDING



Design of 6/8 bit Binary to BCD Converter: Very good detail shows a clear understanding of the design of the 6 bit binary to BCD converter. (4/5). Good detail on the 8 bit device with block diagram shown. (4/5) Re-design to suit new applications: 5KHz design could have been explained more thoroughly (1/3); Mux/Demux again could have been explained more (1/3); 7 segment decoder explained

**15**

satisfactorily (2/3); Full block design shown and understood. (3/6) Total (15/25)

## ANALYSIS



Analysis of Xilinx Vivado simulation results: 6 bit analysis could have been explained more fully. (2/5) 8 bit analysis clear and correct. (4/5) Analysis of adapted application results: Very good set of results obtained which have been analysed well. 5KHz clock (3/5); Mux/Demux (3/5) and 7 segment decoder. (4/5) Total (16/25)

**16**

## PRACTICAL APPLICATION AND DEP.



Downloading and testing to specification on Nexys 4 DDR FPGA board: Very good set of practical results which have been well analysed for the 8 bit device. (7/10) Downloading and testing of new application results on the Nexys 4 DDR FPGA board: Good analysis given on all results obtained. (7/10) Total (14/20)

**14**

## SKILLS FOR PROFESSIONAL PRAC.



Deployment of appropriate media: Good use made of pictorial evidence in showing understanding of assignment results. (3/5) Report layout and readability: All aspects of assignment covered well. (3/5) Appropriate conclusion: Good general conclusion which included some reflection. (4/5) Total (10/15)

**10**