

PSDE5029-Assignment_1-

Emyr_Wyn_Dyer

by Emyr Dyer

FILE	7311_EMYR_DYER_PSDE5029-ASSIGNMENT_1- EMYR_WYN_DYER_155114_1097033473.DOCX (3.98M)		
TIME SUBMITTED	08-FEB-2018 09:30AM (UTC+0000)	WORD COUNT	4345
SUBMISSION ID	82929042	CHARACTER COUNT	29336

HND Electrical & Electronic Engineering

Binary / BCD Conversion in VHDL

PSDE5029 – Virtual Instrumentation and
Control Systems

Author: Emyr Wyn Dyer
Submission Date: 08/02/2018
Word Count: 4345

CONTENTS

1	Introduction	1
2	Research.....	2
2.1	Structure of a typical VHDL file	2
2.2	Binary to BCD conversion.....	4
2.3	IP Block Design in Vivado Design Suite	6
3	Design of assignment requirements	7
3.1	6-bit Binary to BCD Converter	7
3.1.1	Main VHDL File	7
3.1.2	Test Bench File	10
3.1.3	Project Sources	11
3.2	8-bit Binary to BCD Converter.....	12
3.2.1	Main VHDL File	12
3.2.2	Test Bench File	14
3.2.3	Constraints File.....	16
4	Analysis of Xilinx Vivado simulated results	17
4.1	6-bit Binary to BCD Converter Results and Analysis	17
4.2	8-bit Binary to BCD Converter Results and Analysis	18
5	Analysis of Nexys 4 FPGA practical results.....	20
6	Adapting design to suit new purposes or applications.....	22
6.1	5KHz Clock.....	22
6.1.1	Main VHDL File	22
6.1.2	Test Bench File	23
6.2	Mux Demux.....	24
6.2.1	Main VHDL File	24
6.2.2	Test Bench File	25
6.3	BCD to Seven Segment Display	27
6.3.1	Main VHDL File	27
6.3.2	Test Bench File	28
6.3.3	Constraints File.....	30
6.4	Final Build.....	30
6.4.1	Main VHDL File	30
6.4.2	Constraints File.....	33
7	Analysis of Xilinx Vivado simulated results	34

7.1	5KHz Clock Results and Analysis	34
7.2	Seven Segment Display Results and Analysis.....	34
8	Analysis of Nexys 4 DDR FPGA practical results.....	36
9	Conclusion.....	38
10	Works Cited.....	39
11	Image Sources	40

Figures

Figure 2.1: Clock Frequency Conversion Code.....	2
Figure 2.2: Part 1 of Test Bench	3
Figure 2.3: Part 2 of Test Bench	3
Figure 2.4: Binary to BCD Function Table	4
Figure 2.5: 8-bit Binary to BCD connection.....	5
Figure 2.6: Typical Block Design in Vivado	6
Figure 3.1: Binary to BCD 6-bit VHDL Part 1	7
Figure 3.2: Binary to BCD 6-bit VHDL Part 2	7
Figure 3.3: Binary to BCD 6-bit VHDL Part 3	8
Figure 3.4: Binary to BCD 6-bit VHDL Part 4	9
Figure 3.5: 6-bit Binary to BCD Test Bench VHDL Part 1.....	10
Figure 3.6: 6-bit Binary to BCD Test Bench VHDL Part 2.....	10
Figure 3.7: 6-bit Binary to BCD Test Bench VHDL Part 3.....	11
Figure 3.8: 6-bit Binary to BCD Sources	11
Figure 3.9: 8-bit Binary to BCD Block Diagram.....	12
Figure 3.10: 8-bit Binary to BCD VHDL Part 1	12
Figure 3.11: 8-bit Binary to BCD VHDL Part 2	13
Figure 3.12: 8-bit Binary to BCD VHDL Part 3	13
Figure 3.13: 8-bit Binary to BCD Test Bench VHDL Part 1.....	14
Figure 3.14: 8-bit Binary to BCD Test Bench VHDL Part 2.....	14
Figure 3.15: 8-bit Binary to BCD Test Bench VHDL Part 3.....	15
Figure 3.16: 8-bit Binary to BCD Test Bench VHDL Part 4.....	15
Figure 3.17: 8-bit Binary to BCD Constraints File.....	16
Figure 4.1: Binary Value of 28/29 identified	17
Figure 4.2: Binary Value of 14/15 identified	17
Figure 4.3: Binary Value of 54/55 identified	18
Figure 4.4: Binary Value of 48 identified	18

Figure 4.5: Binary Value of 84 identified	19
Figure 4.6: Binary Value of 129 identified	19
Figure 5.1: 8-bit Binary to BCD Converter on FPGA outputting value of 12	20
Figure 5.2: 8-bit Binary to BCD Converter on FPGA outputting value of 42	20
Figure 5.3: 8-bit Binary to BCD Converter on FPGA outputting value of 129	21
Figure 6.1: 5KHz Clock VHDL file	22
Figure 6.2: 5KHz Clock Test Bench File Part 1	23
Figure 6.3: 5KHz Clock Test Bench File Part 2	23
Figure 6.4: Mux Demux VHDL File Part 1	24
Figure 6.5: Mux Demux VHDL File Part 2	25
Figure 6.6: Mux Demux Test Bench File Part 1	25
Figure 6.7: Mux Demux Test Bench File Part 2	26
Figure 6.8: Mux Demux Test Bench File Part 3	26
Figure 6.9: BCD to Seven Segment Display VHDL File Part 1	27
Figure 6.10: BCD to Seven Segment Display VHDL File Part 2	28
Figure 6.11: BCD to Seven Segment Display Test Bench File Part 1	28
Figure 6.12: BCD to Seven Segment Display Test Bench File Part 2	29
Figure 6.13: BCD to Seven Segment Display Test Bench File Part 3	29
Figure 6.14: BCD to Seven Segment Display Constraints File	30
Figure 6.15: Final Build VHDL Block Diagram View	30
Figure 6.16: Final Build VHDL File Part 1	31
Figure 6.17: : Final Build VHDL File Part 2	31
Figure 6.18: : Final Build VHDL File Part 3	31
Figure 6.19: : Final Build VHDL File Part 4	32
Figure 6.20: Final Build VHDL File Part 5	32
Figure 6.21: Final Build VHDL File Part 6	33
Figure 6.22: Final Build VHDL Constraints File	33
Figure 7.1: Measured at 4ms	34
Figure 7.2: Proof of '0' where sg is turned off/1.....	34
Figure 7.3: Proof that Display turns off when Binary value greater than 1001 is sent.....	35
Figure 8.1: Nexys 4 FPGA displaying 12	36
Figure 8.2: Nexys 4 FPGA displaying 42	36
Figure 8.3: Nexys 4 FPGA displaying 129	37

1 INTRODUCTION

The overall goal of this report is show an understanding of how VHDL and FPGAs can be used to effectively replicate existing electronic designs and adapt them for use in other designs. This understanding will be found using Vivado, which is a coding application designed to program FPGAs using either VHDL or Verilog. Through the development process of the code, testing will be carried out and documented as well as discussed to explain the functionality of the VHDL code before further implementation.

Very brief Introduction... What do the following terms mean: VHDL; FPGA?

2 RESEARCH

2.1 STRUCTURE OF A TYPICAL VHDL FILE

Unlike nearly all other programming languages, VHDL is a 'hardware based' language. (Mealy, 2017) The term 'hardware based' does not mean referencing only Inputs and Outputs of a system, but rather emulating hardware in its entirety, effectively building and implementing hardware through software. The code written in this investigation, will mostly follow the 'Behavioural architecture' which relates to how the designed circuit will react to Inputs fed in. How the code really works is defined in the 'Process' element of the code. The 'Process' is run sequentially with Inputs taken in, and sequential statements carrying out processes on those Inputs such as logical or mathematical, to provide a desired Output (Readler, 2014). The statements used for this task are commonly IF/ELSEIF as well as CASE statements, which are also used in other languages. The actual method of implementation of the code is handled during synthesis. In the style used above to code a 'Behavioural architecture' design, the following elements are featured:

Element	Function/Characteristic
Code Description	Not essential to operation, but important as it documents the intended function of the code as well as modifications done to the code.
Libraries	References to pre-defined IPs that can be implemented in the design.
Entity	Defines the Inputs and Outputs of the VHDL design
Behavioural Architecture	As well as containing the Process, declarations for any signals (variables) are included at the head of this section.
Process	Here, the signals and Inputs are taken in, processed using statements, and then Outputted.

```
-- Hertz100.vhdl      A description of the code functionality at the head of
--                   the file which can give a brief description of how the
--                   code works as well information regarding owners of the
--                   code, iteration version and any changes from previous
--                   version.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;      --The IEEE library is essential to coding VHDL
use IEEE.NUMERIC_STD.ALL;         --The following three library elements are used
use IEEE.std_logic_unsigned.all;  --in the following code and are brought in by
                                   --IEEE library.

entity Hertz100 is
    Port ( CLK100MHZ : in STD_LOGIC;  --In the entity Hertz100, the I/O are declared in
          Rst : in std_logic;          --relation to the FPGA.
          Hz100 : out STD_LOGIC
        );
end Hertz100;                    --The entity listing ends.

architecture Behavioral of Hertz100 is
    --Initial declarations for the code are included
    --in the Behavioral architecture of the code.
    signal cnt1 : integer := 0;      --The following integer values initialized.
    signal temp1 : std_logic := '0';
    begin
        process(CLK100MHZ,Rst)      --The process or functionality of the VHDL file is
        --coded below and the two inputs are fed into the
        --process.
        begin
            if Rst = '1' then
                temp1 <= '0';
                cnt1 <= 0;
            elsif rising_edge (CLK100MHZ) then
                if(cnt1= 500000) then
                    temp1 <= not temp1;
                    cnt1 <= 0;
                else
                    cnt1 <= cnt1 +1 ;
                end if;
            end if;
        end process;
        Hz100 <= temp1;              --The process ends
        --The product of the above process is stored before
        --looping.
    end Behavioral;                 --The Behavioral architecture ends.
```

Figure 2.1: Clock Frequency Conversion Code

The other type of VHDL file used in conjunction with the designed VHDL file is a 'Test Bench' file, which is usually named the same as the VHDL design file and indicated with a suffix at the end of the file of '_tb'. The purpose of a Test Bench file is to apply Input signals to the design, to simulate the actual Inputs when the VHDL file is synthesised and uploaded to a FPGA. The Architecture of the code somewhat mirrors the design code, only instead of logic, code is written stating which Inputs have a signal applied, in which sequence and for how long. The aim of this method of testing is to simulate every combination of Input that can be applied to the design, and the Output can be monitored to find out if the design performs to expectations.

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

-- Uncomment the following library declaration if instantiating
entity Hertz100_tb is
-- Port ( );
end Hertz100_tb;

architecture Behavioral of Hertz100_tb is

Component Hertz100
Port ( CLK100MHZ      : in STD_LOGIC;
      Rst             : in STD_LOGIC;
      Hz100          : out STD_LOGIC
      );
end Component;

--Inputs
signal CLK100MHZ : STD_LOGIC := '0';
signal Rst       : STD_LOGIC := '1';

--Outputs
signal Hz100 : STD_LOGIC;

--Clock definitions
constant CLK100MHZ_period : time := 10ns;
```

Figure 2.2: Part 1 of Test Bench

```
begin

--Instantiate the UUT
    UUT : Hertz100

    Port Map ( CLK100MHZ => CLK100MHZ,
              Rst        => Rst,
              Hz100      => Hz100
              );

--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;
        Rst <= '0';
        wait for 40ms;
        Rst <= '1';
        wait;
    end process;

end Behavioral;
```

Figure 2.3: Part 2 of Test Bench

Good detail given.

2.2 BINARY TO BCD CONVERSION

From the Datasheet for the DM74184/DM74185A BCD/Binary Converter (National Semiconductor, 2014), information about the functionality of the production chips, their Inputs and Outputs, and their limitations, be it operating conditions or electrical characteristics. Also included are typical configurations of the chip to cater to a specific design goal, and documentation detailing the principles of operation for the chip, such as Function Tables which describe which voltage Outputs are produced by the chip depending on voltage states on the Inputs.

Function Tables													
Binary Words	Inputs						Outputs						
	E	Binary Select				Enable G	Y8	Y7	Y6	Y5	Y4	Y3	Y2
1	L	L	L	L	L	L	H	H	L	L	L	L	L
3	L	L	L	L	H	L	H	H	L	L	L	L	L
5	L	L	L	H	L	L	H	H	L	L	L	L	H
7	L	L	L	H	H	L	H	H	L	L	L	L	H
9	L	L	H	L	L	L	H	H	L	L	L	H	L
11	L	L	H	L	H	L	H	H	L	L	H	L	L
13	L	L	H	H	L	L	H	H	L	L	H	L	L
15	L	L	H	H	H	L	H	H	L	L	H	L	H
17	L	H	L	L	L	L	H	H	L	L	H	L	H
19	L	H	L	L	H	L	H	H	L	L	H	H	L
21	L	H	L	H	L	L	H	H	L	H	L	L	L
23	L	H	L	H	H	L	H	H	L	H	L	L	L
25	L	H	H	L	L	L	H	H	L	H	L	L	H
27	L	H	H	L	H	L	H	H	L	H	L	L	H
29	L	H	H	H	L	L	H	H	L	H	L	H	L
31	L	H	H	H	H	L	H	H	L	H	H	L	L
33	H	L	L	L	L	L	H	H	L	H	H	L	L
35	H	L	L	L	H	L	H	H	L	H	H	L	H
37	H	L	L	H	L	L	H	H	L	H	H	L	H
39	H	L	L	H	H	L	H	H	L	H	H	H	L
41	H	L	H	L	L	L	H	H	H	L	L	L	L
43	H	L	H	L	H	L	H	H	H	L	L	L	L
45	H	L	H	H	L	L	H	H	H	L	L	L	H
47	H	L	H	H	H	L	H	H	H	L	L	L	H
49	H	H	L	L	L	L	H	H	H	L	L	H	L
51	H	H	L	L	H	L	H	H	H	L	H	L	L
53	H	H	L	H	L	L	H	H	H	L	H	L	L
55	H	H	L	H	H	L	H	H	H	L	H	L	H
57	H	H	H	L	L	L	H	H	H	L	H	L	H
59	H	H	H	L	H	L	H	H	H	L	H	H	L
61	H	H	H	H	L	L	H	H	H	H	L	L	L
63	H	H	H	H	H	L	H	H	H	H	L	L	L

Figure 2.4: Binary to BCD Function Table

In Figure 2.4, The Outputs according to Inputs are shown and from this, it can be interpreted how the design would function. Where the Outputs are named Y1 to Y6, in the design Y1 to Y3 would Output on the design as U1 to U3 to represent a Unit digit (an external U0 is used to represent binary 1), Y4 to Y6 would Output on the design as T0 to T3, to provide the maximum binary number range, which is 0 to 63. To produce a Binary to BCD design that can count from 0 to 255, it is shown in the Datasheet that it possible to build an 8-bit Binary to BCD converter from 3 of the 6-bit versions as shown in Figure 2.5.

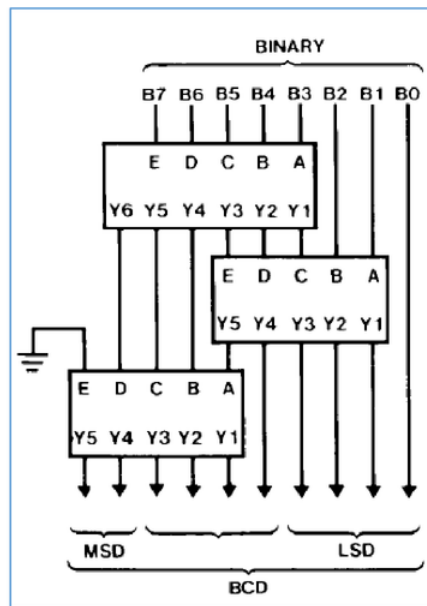


Figure 2.5: 8-bit Binary to BCD connection

Very good detail.

2.3 IP BLOCK DESIGN IN VIVADO DESIGN SUITE

To create a larger Intellectual Property, the manner of linking the individual IPs together is done in the Block Design view of Vivado. Here IPs can be imported and linked to other IPs to pass Output signals to Inputs as required. Also, in design view it is possible to place Input and Output Ports which refer to the I/O on the FPGA, that the re-packaged IP is to be uploaded to. To tie the Ports of the IP to the I/O of the FPGA, a Constraints file is imported to the Design Project in Vivado to match the Virtual Ports to the Physical Ports on the FPGA.

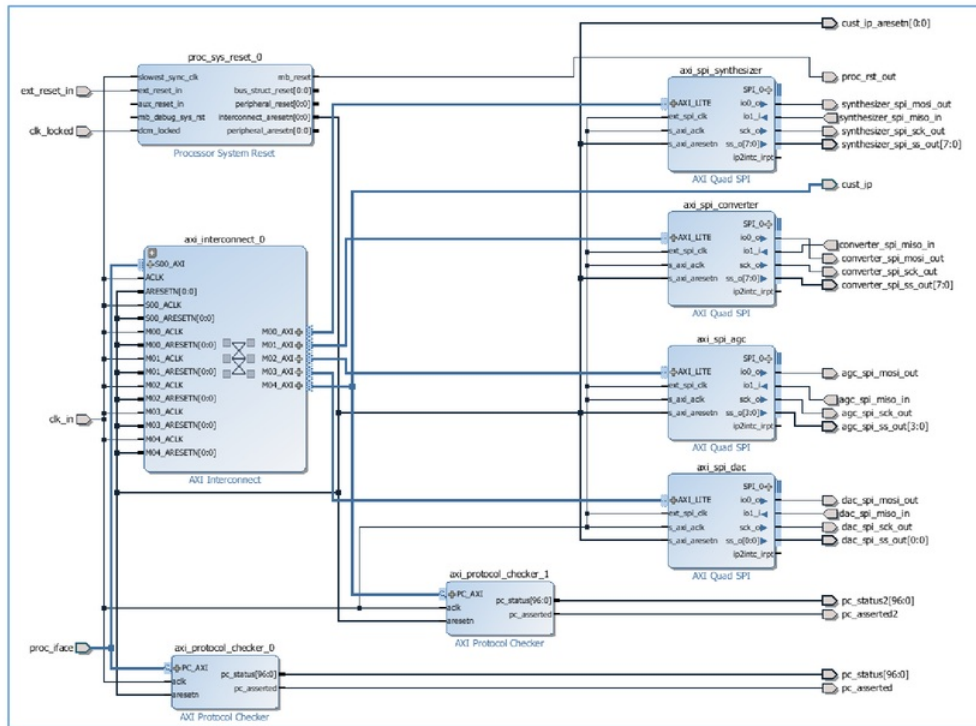


Figure 2.6: Typical Block Design in Vivado

This diagram may be typical but has no relevance to you showing understanding of how IP blocks are used

3 DESIGN OF ASSIGNMENT REQUIREMENTS

3.1 6-BIT BINARY TO BCD CONVERTER

By looking back at the datasheet for the DM74184 and DM74185A chip and examining the function table which defined the Output states for the binary Inputs, it was possible to build a VHDL version of the chip. *Very good*

3.1.1 Main VHDL File

The Inputs and Outputs were defined at the head of the code in the entity of the VHDL file.

```
22 : library IEEE;
23 : use IEEE.STD_LOGIC_1164.ALL;
24
25 entity bcd_6_bit is
26
27     Port ( A,B,C,D,E : in STD_LOGIC;
28           Y1,Y2,Y3,Y4,Y5,Y6 : out STD_LOGIC);
29
30 end bcd_6_bit;
```

Figure 3.1: Binary to BCD 6-bit VHDL Part 1

The internal signals of the architecture were then defined with the Input signal (Inp) carrying 5 bits (4 down to 0) and the Output signal (Q) carrying 6 bits (5 down to 0). At the beginning, the states of Inputs A to E are applied to the 5 bits of the Input signal. Next the 'Inp' is sequentially checked using a case statement, where if the condition of the current 'Inp' state is matched, then 'Q' is set accordingly, so that if a binary value of "00100" is put through, then Q is set to "000100".

```
32 architecture Behavioral of bcd_6_bit is
33 : signal Inp : std_logic_vector (4 downto 0);
34 : signal Q : std_logic_vector (5 downto 0);
35
36 : begin
37
38 :     Inp(0) <= A;
39 :     Inp(1) <= B;
40 :     Inp(2) <= C;
41 :     Inp(3) <= D;
42 :     Inp(4) <= E;
43
44 : process (Inp)
45 : begin
46 : case Inp is
47 : WHEN "00000" => -- 0/1
48 : Q <= ("000000");
49 : WHEN "00001" => -- 2/3
50 : Q <= ("000001");
51 : WHEN "00010" => -- 4/5
52 : Q <= ("000010");
53 : WHEN "00011" => -- 6/7
54 : Q <= ("000011");
55 : WHEN "00100" => -- 8/9
56 : Q <= ("000100");
57 : WHEN "00101" => -- 10/11
58 : Q <= ("001000");
59 : WHEN "00110" => -- 12/13
60 : Q <= ("001001");
61 : WHEN "00111" => -- 14/15
62 : Q <= ("001010");
63 : WHEN "01000" => -- 16/17
64 : Q <= ("001011");
65 : WHEN "01001" => -- 18/19
66 : Q <= ("001100");
67 : WHEN "01010" => -- 20/21
```

Figure 3.2: Binary to BCD 6-bit VHDL Part 2

Good

Every entry in the function table from the datasheet is coded into the VHDL file.

```
69 WHEN "01011" => -- 22/23
70   Q <= ("010001");
71 WHEN "01100" => -- 24/25
72   Q <= ("010010");
73 WHEN "01101" => -- 26/27
74   Q <= ("010011");
75 WHEN "01110" => -- 28/29
76   Q <= ("010100");
77 WHEN "01111" => -- 30/31
78   Q <= ("011000");
79 WHEN "10000" => -- 32/33
80   Q <= ("011001");
81 WHEN "10001" => -- 34/35
82   Q <= ("011010");
83 WHEN "10010" => -- 36/37
84   Q <= ("011011");
85 WHEN "10011" => -- 38/39
86   Q <= ("011100");
87 WHEN "10100" => -- 40/41
88   Q <= ("100000");
89 WHEN "10101" => -- 42/43
90   Q <= ("100001");
91 WHEN "10110" => -- 44/45
92   Q <= ("100010");
93 WHEN "10111" => -- 46/47
94   Q <= ("100011");
95 WHEN "11000" => -- 48/49
96   Q <= ("100100");
97 WHEN "11001" => -- 50/51
98   Q <= ("101000");
99 WHEN "11010" => -- 52/53
.00   Q <= ("101001");
.01 WHEN "11011" => -- 54/55
.02   Q <= ("101010");
```

Figure 3.3: Binary to BCD 6-bit VHDL Part 3

At the end of the case statement, the process is also ended and the signal Q, is applied to the outgoing ports.

```
90 Q <= ("100001");
91 WHEN "10110" => -- 44/45
92 Q <= ("100010");
93 WHEN "10111" => -- 46/47
94 Q <= ("100011");
95 WHEN "11000" => -- 48/49
96 Q <= ("100100");
97 WHEN "11001" => -- 50/51
98 Q <= ("101000");
99 WHEN "11010" => -- 52/53
100 Q <= ("101001");
101 WHEN "11011" => -- 54/55
102 Q <= ("101010");
103 WHEN "11100" => -- 56/57
104 Q <= ("101011");
105 WHEN "11101" => -- 58/59
106 Q <= ("101100");
107 WHEN "11110" => -- 60/61
108 Q <= ("110000");
109 WHEN "11111" => -- 62/63
110 Q <= ("110001");
111 WHEN others => --
112 Q <= ("000000");
113
114 end case;
115 end process;
116
117 Y1 <= Q(0);
118 Y2 <= Q(1);
119 Y3 <= Q(2);
120 Y4 <= Q(3);
121 Y5 <= Q(4);
122 Y6 <= Q(5);
123
124 end behavioral;
```

Figure 3.4: Binary to BCD 6-bit VHDL Part 4

Good explanation given.

3.1.2 Test Bench File

To ensure that the written VHDL works, a test bench file is written to simulate the Inputs to go through every condition in the case statement of the 6-bit Binary to BCD VHDL. After the definition of the entity, the Behavioural is established where the Unit Under Test is identified as well as the Input and Output Ports to the VHDL file.

```
1  : library IEEE;
2  : use IEEE.STD_LOGIC_1164.ALL;
3  entity bcd_6_bit_tb is
4  : -- Port ( );
5  end bcd_6_bit_tb;
6  architecture Behavioral of bcd_6_bit_tb is
7  : -- Component Declaration for the Unit Under Test (UUT)
8  component bcd_6_bit
9  : Port ( A,B,C,D,E : in std_logic;
10 : Y1,Y2,Y3,Y4,Y5,Y6 : out std_logic
11 : );
12 end component;
```

Figure 3.5: 6-bit Binary to BCD Test Bench VHDL Part 1

Input and Output signals are established, with the Input signals initialised at '0'. Clock time periods are also defined to trigger the inputs in a fashion that makes the output logically comprehensible. In the PORT MAP, the Inputs and Outputs of the UUT and the TB are tied.

```
--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: bcd_6_bit PORT MAP (
A => A,
B => B,
C => C,
D => D,
E => E,
Y1 => Y1,
Y2 => Y2,
Y3 => Y3,
Y4 => Y4,
Y5 => Y5,
Y6 => Y6
);
```

Figure 3.6: 6-bit Binary to BCD Test Bench VHDL Part 2

The clock process is further defined so that to correctly trigger each input bit, alternating between 0 and 1 to produce the 64 binary values.

```

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

```

Figure 3.7: 6-bit Binary to BCD Test Bench VHDL Part 3

3.1.3 Project Sources

The Relationship between the files described in Section 3.1 are shown in Figure 3.8. With all files written, and then tested, IP packaging is carried out for importing into other VHDL projects.

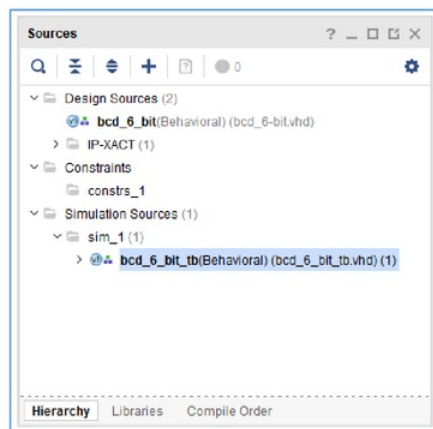


Figure 3.8: 6-bit Binary to BCD Sources

The 6-bit Binary to BCD Converter must now be converted to an 8-bit version to produce a binary range of 0 to 255.

The 6-bit Binary to BCD Converter is imported through the IP Catalog where the directory location of the relevant IP resources is identified. By going into the Block Diagram view by clicking 'Generate Block Diagram', the 6-bit Converter IP was imported and configured as was shown in Figure 2.5, with Virtual Ports established, and then Wrapped before an IP packaging was carried out.



```

10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.VCOMPONENTS.ALL;
14 entity design_1_wrapper is
15     port (
16         B0 : in STD_LOGIC;
17         B1 : in STD_LOGIC;
18         B2 : in STD_LOGIC;
19         B3 : in STD_LOGIC;
20         B4 : in STD_LOGIC;
21         B5 : in STD_LOGIC;
22         B6 : in STD_LOGIC;
23         B7 : in STD_LOGIC;
24         H0 : out STD_LOGIC;
25         H1 : out STD_LOGIC;
26         SW15 : in STD_LOGIC;
27         T0 : out STD_LOGIC;
28         T1 : out STD_LOGIC;
29         T2 : out STD_LOGIC;
30         T3 : out STD_LOGIC;
31         Un0 : out STD_LOGIC;
32         Un1 : out STD_LOGIC;
33         Un2 : out STD_LOGIC;
34         Un3 : out STD_LOGIC
35     );
36 end design_1_wrapper;

```

Figure 3.10: 8-bit Binary to BCD VHDL Part 1

```

architecture STRUCTURE of design_1_wrapper is
  component design_1 is
    port (
      SW15 : in STD_LOGIC;
      B0 : in STD_LOGIC;
      Un0 : out STD_LOGIC;
      Un1 : out STD_LOGIC;
      Un2 : out STD_LOGIC;
      Un3 : out STD_LOGIC;
      T0 : out STD_LOGIC;
      T1 : out STD_LOGIC;
      T2 : out STD_LOGIC;
      T3 : out STD_LOGIC;
      H0 : out STD_LOGIC;
      H1 : out 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
    );
  end component design_1;

```

Figure 3.11: 8-bit Binary to BCD VHDL Part 2

```

62 : begin
63 : design_1_i: component design_1
64 :   port map (
65 :     B0 => B0,
66 :     B1 => B1,
67 :     B2 => B2,
68 :     B3 => B3,
69 :     B4 => B4,
70 :     B5 => B5,
71 :     B6 => B6,
72 :     B7 => B7,
73 :     H0 => H0,
74 :     H1 => H1,
75 :     SW15 => SW15,
76 :     T0 => T0,
77 :     T1 => T1,
78 :     T2 => T2,
79 :     T3 => T3,
80 :     Un0 => Un0,
81 :     Un1 => Un1,
82 :     Un2 => Un2,
83 :     Un3 => Un3
84 :   );
85 : end STRUCTURE;

```

Figure 3.12: 8-bit Binary to BCD VHDL Part 3

3.2.2 Test Bench File

As before, the entity is defined along with the behavioural with the Ports used matching the constraints file (shown in Section 3.2.3).

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5
6  entity Bin_BCD_8bit_tb is
7  -- Port ( );
8  end Bin_BCD_8bit_tb;
9
10 architecture Behavioral of Bin_BCD_8bit_tb is
11
12  -- Component Declaration for the Unit Under Test (UUT)
13
14  COMPONENT design_1_wrapper
15  Port ( B0,B1,B2,B3,B4,B5,B6,B7,SW15      : in std_logic;
16        Un0,Un1,Un2,Un3,T0,T1,T2,T3,H0,H1  : out std_logic
17        );
18  END COMPONENT;
```

Figure 3.13: 8-bit Binary to BCD Test Bench VHDL Part 1

This time, instead of using a clock to check through every permutation of the circuit, select values are chosen for comparison to by selecting a value from between 0 and 255, then figuring out the BCD value to be expected before carrying out the Analysis. Although an initial value of 170 was chosen the value was then altered to 48.

```
21  --Inputs
22
23  signal B0 : std_logic := '0';
24  signal B1 : std_logic := '1';
25  signal B2 : std_logic := '0';
26  signal B3 : std_logic := '1';
27  signal B4 : std_logic := '0';
28  signal B5 : std_logic := '1';
29  signal B6 : std_logic := '0';
30  signal B7 : std_logic := '1';
31  -- First test @ 170
32  signal SW15 : std_logic := '0';
33
34  --Outputs
35
36  signal Un0 : std_logic;
37  signal Un1 : std_logic;
38  signal Un2 : std_logic;
39  signal Un3 : std_logic;
40  signal T0 : std_logic;
41  signal T1 : std_logic;
42  signal T2 : std_logic;
43  signal T3 : std_logic;
44  signal H0 : std_logic;
45  signal H1 : std_logic;
46
47  --Clocks
48  -- No Clocks
49
```

Figure 3.14: 8-bit Binary to BCD Test Bench VHDL Part 2

The Test Bench values are applied to the Wrapped 8-bit Converter.

```

50 begin
51
52     -- Instantiate the Unit Under Test (UUT)
53     uut: design_1_wrapper PORT MAP (
54         B0 => B0,
55         B1 => B1,
56         B2 => B2,
57         B3 => B3,
58         B4 => B4,
59         B5 => B5,
60         B6 => B6,
61         B7 => B7,
62         SW15 => SW15,
63         Un0 => Un0,
64         Un1 => Un1,
65         Un2 => Un2,
66         Un3 => Un3,
67         T0 => T0,
68         T1 => T1,
69         T2 => T2,
70         T3 => T3,
71         H0 => H0,
72         H1 => H1
73     );

```

Figure 3.15: 8-bit Binary to BCD Test Bench VHDL Part 3

The other pre-chosen values of 131 and 195 are then applied to the Wrapped 8-bit converter, though were later changed to 84 and 129 to cover a better range.

```

75 -- Clock process definitions
76
77 -- Stimulus process
78 stim_proc: process
79 begin
80
81     wait for 1ms;
82     -- Second test @ 131
83     B0 <= '1';
84     B1 <= '1';
85     B2 <= '0';
86     B3 <= '0';
87     B4 <= '0';
88     B5 <= '0';
89     B6 <= '0';
90     B7 <= '1';
91     wait for 1ms;
92     -- Third test @ 195
93     B0 <= '1';
94     B1 <= '1';
95     B2 <= '0';
96     B3 <= '0';
97     B4 <= '0';
98     B5 <= '0';
99     B6 <= '1';
100    B7 <= '1';
101
102    wait;
103 end process;
104
105 end Behavioral;

```

Figure 3.16: 8-bit Binary to BCD Test Bench VHDL Part 4

3.2.3 Constraints File

It is this file that links Virtual Port Names, to Actual Port Names on the FPGA.

```
1 ##Switches
2
3 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { B0 }]; #IO_L24N_T3_RS0_15 Sch=sv[0]
4 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { B1 }]; #IO_L3N_T0_D03_EMCCLK_14 Sch=sv[1]
5 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { B2 }]; #IO_L6N_T0_D09_VREF_14 Sch=sv[2]
6 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { B3 }]; #IO_L13N_T2_MRCC_14 Sch=sv[3]
7 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { B4 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
8 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { B5 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
9 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { B6 }]; #IO_L17N_T2_A13_D29_14 Sch=sv[6]
10 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { B7 }]; #IO_L6N_T0_D07_14 Sch=sv[7]
11 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { SW15 }]; #IO_L21P_T3_D05_14 Sch=sv[15]
12
13 ##LED's
14
15 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Un0 }]; #IO_L18P_T2_A24_15 Sch=led[0]
16 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { Un1 }]; #IO_L24P_T3_RS1_15 Sch=led[1]
17 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { Un2 }]; #IO_L17N_T2_A25_15 Sch=led[2]
18 set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { Un3 }]; #IO_L8P_T1_D11_14 Sch=led[3]
19 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { T0 }]; #IO_L7P_T1_D09_14 Sch=led[4]
20 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { T1 }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
21 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { T2 }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
22 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { T3 }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
23 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { H0 }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
24 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { H1 }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
25
```

Figure 3.17: 8-bit Binary to BCD Constraints File

*Excellent detail shown in the design and simulation
of the 8 bit device*

4 ANALYSIS OF XILINX VIVADO SIMULATED RESULTS

Below the Tests for the 6-bit Binary to BCD Converter and the 8-bit Binary to BCD Converter are shown and Analysed for performance.

4.1 6-BIT BINARY TO BCD CONVERTER RESULTS AND ANALYSIS

By comparing to the function table procured from the Datasheet (Figure 2.4), select values were chosen and matched in the simulation. All three results showed correct behaviour for the design as the high peaks or binary 1s corresponded to the H symbol from the table, with 0s corresponding to L. With the successful test, the design was final and packaged as an IP to be used in future designs.

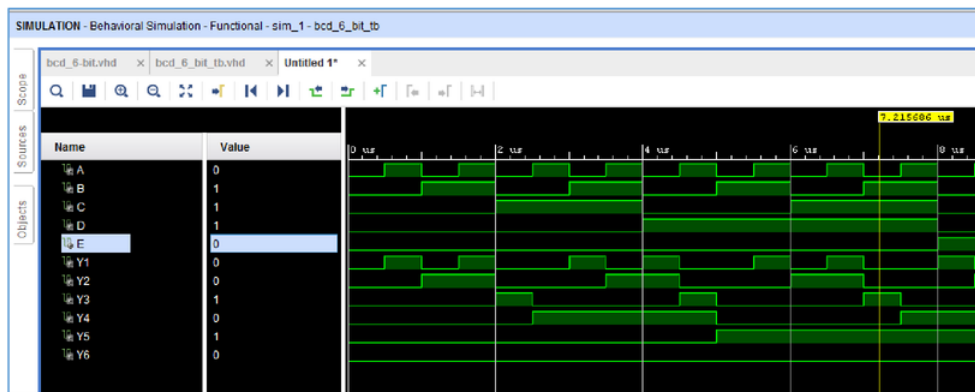


Figure 4.1: Binary Value of 28/29 identified

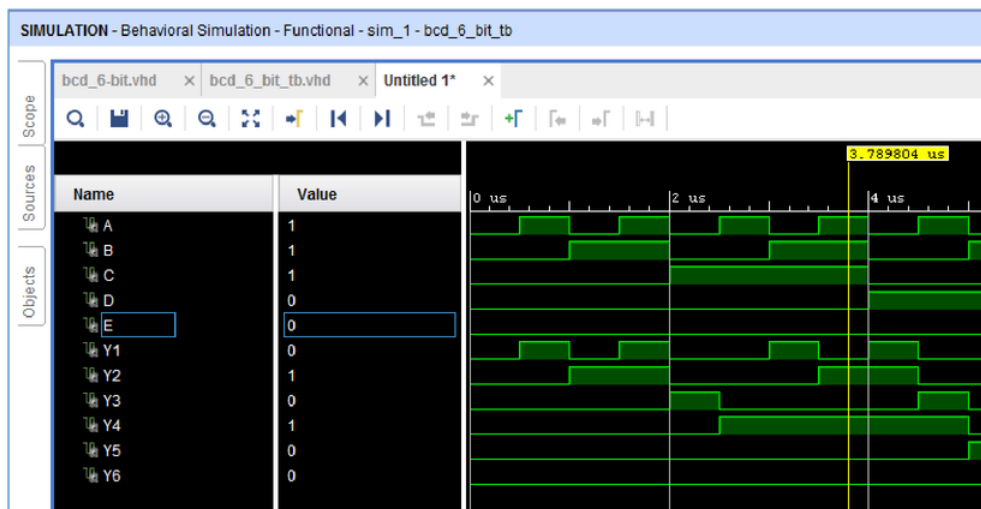


Figure 4.2: Binary Value of 14/15 identified

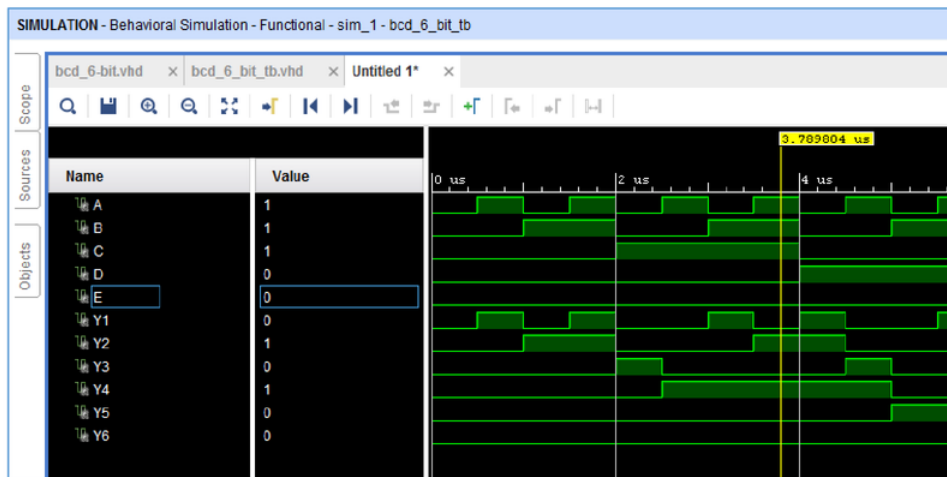


Figure 4.3: Binary Value of 54/55 identified
[Bit more detail on analysis please](#)

4.2 8-BIT BINARY TO BCD CONVERTER RESULTS AND ANALYSIS

The values set to the Test Bench file were reflected in the simulated results for the circuit. This has shown that the production of an IP by importing another IP has been successful. In theory this can be carried further by using this packaged IP in an even larger IP project, however before this is done, a practical test should be carried out.

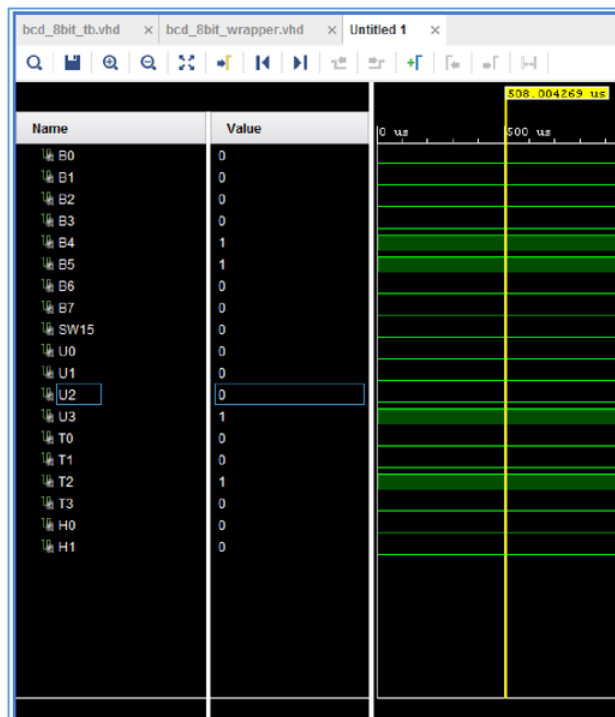


Figure 4.4: Binary Value of 48 identified

[Use the simulated results to show your understanding of how the device works!](#)

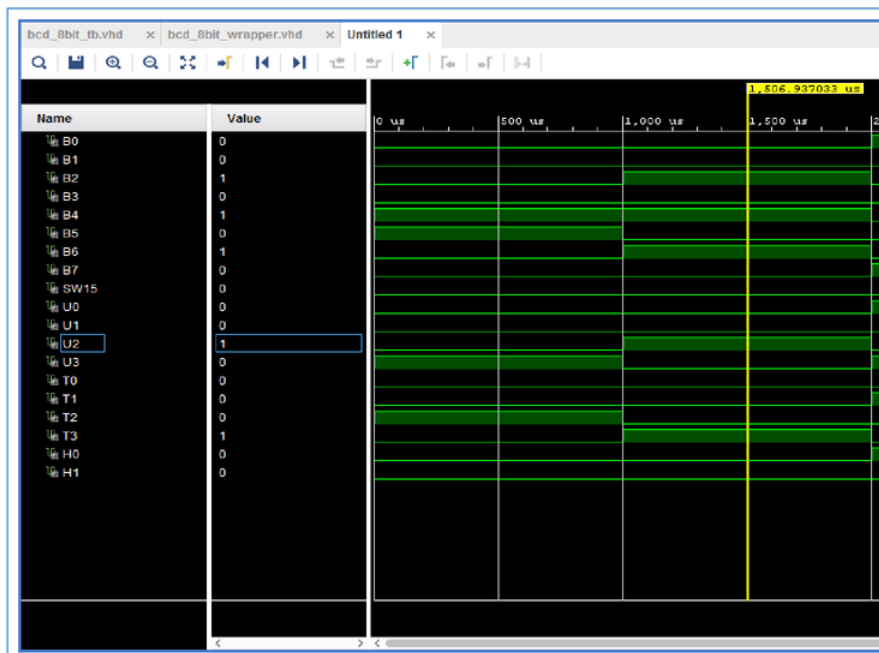


Figure 4.5: Binary Value of 84 identified

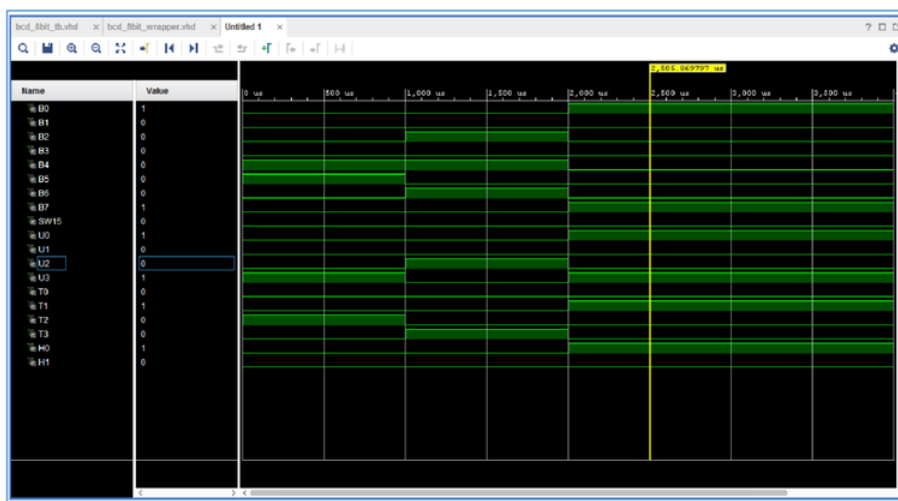


Figure 4.6: Binary Value of 129 identified

5 ANALYSIS OF NEXYS 4 FPGA PRACTICAL RESULTS

By setting the switches on the board to 12, 42, and then 129, images were taken showing the relevant LEDs lighting up, in the case of Figure 2.1 to show the number 12, LED K15 (U2) and LED R17 (T1) lit up. The other two tests showed similar corresponding results as recorded in Figure 5.2 and Figure 5.3. Ideally, I should have used the same values in the simulation as well as the practical test, however the results in the end were positive.

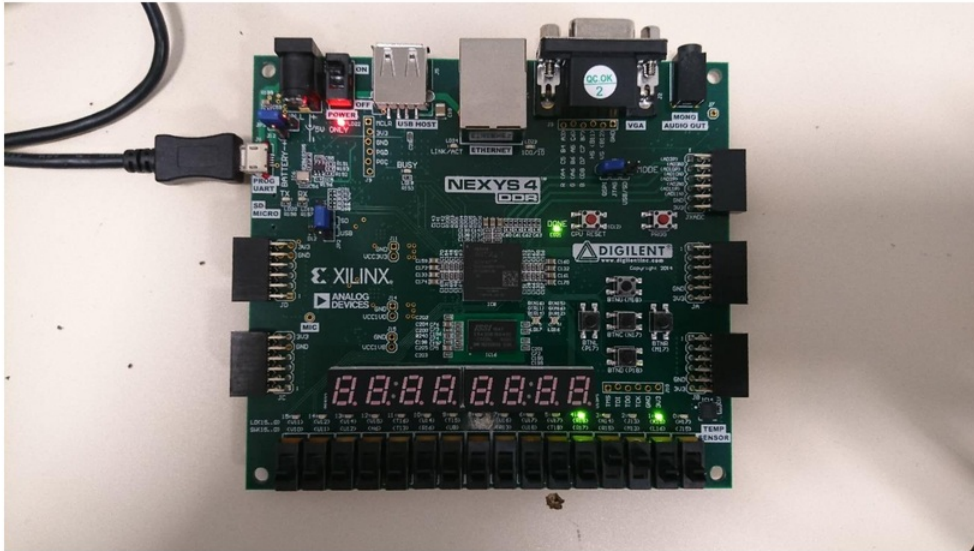


Figure 5.1: 8-bit Binary to BCD Converter on FPGA outputting value of 12

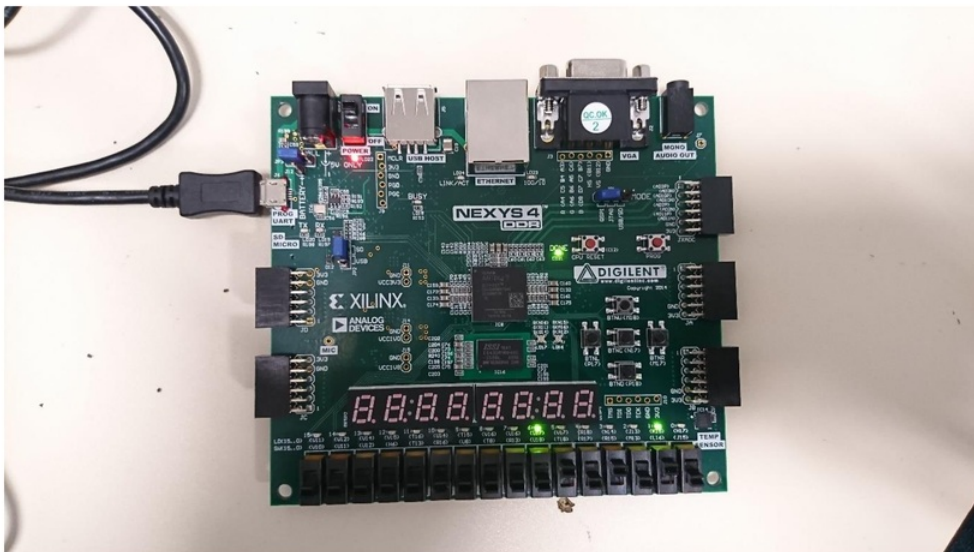


Figure 5.2: 8-bit Binary to BCD Converter on FPGA outputting value of 42

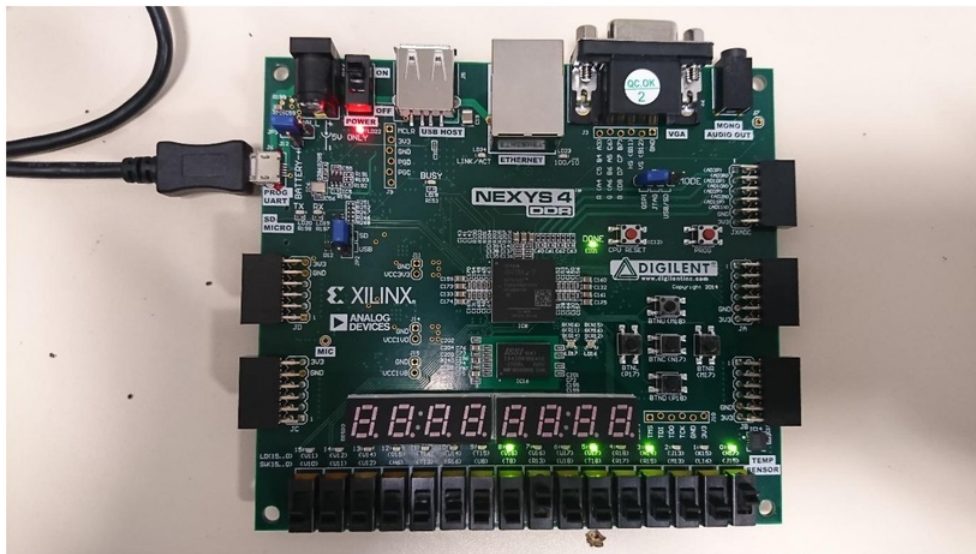


Figure 5.3: 8-bit Binary to BCD Converter on FPGA outputting value of 129

Good use made of pictorial evidence.

6 ADAPTING DESIGN TO SUIT NEW PURPOSES OR APPLICATIONS

While outputting a binary digit entered via switches to be represented on LEDs was successful, it would be better to display a numeric digit on a screen instead, so other IPs were developed using what has been learned about BCD. Elements that would be required include a 5KHz Clock, a Mux/Demux and a BCD to Seven Segment Display.

6.1 5KHz CLOCK

To for a practical test to work, there needs to be a drop down in frequency for the clock. The reason for this is that at 100MHz, which is the default Clock speed of the Nexys 4 board, the Seven Segment Display too fast for a human to see, so in effect all a human would see on the display, is 8. To achieve a slower, more reasonable Clock Frequency, a VHDL file is written.

6.1.1 Main VHDL File

In the entity, the Default 100MHz Clock speed is taken in as an input as well as the Rst, and the 5KHz Clock to be outputted is set as an output. In the behavioural, a count is declared along with a temporary count. The process has many if and else statements that use the count and temporary count to loop 10000 times on the rising edge of the 100MHz clock signal, and once a count of 10000 is reached a pulse is put out through the FiveKHz output signal. There is also a Rst condition which resets both the count and temporary count when the Rst is triggered.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  entity Clock_5KHz is
6  Port ( CLK100MHZ : in STD_LOGIC;
7        Rst : in std_logic;
8        FiveKHz : out STD_LOGIC
9  );
10 end Clock_5KHz;
11 architecture Behavioral of Clock_5KHz is
12 signal cnt1 : integer := 0;
13 signal templ : std_logic := '0';
14 begin
15 process (CLK100MHZ,Rst)
16 begin
17 if Rst = '1' then
18 templ <= '0';
19 cnt1 <= 0;
20 elsif rising_edge (CLK100MHZ) then
21 if(cnt1= 10000) then
22 templ <= not templ;
23 cnt1 <= 0;
24 else
25 cnt1 <= cnt1 +1 ;
26 end if;
27 end if;
28 end process;
29 FiveKHz <= templ;
30
31 end Behavioral;
```

Design not explained well.

Figure 6.1: 5KHz Clock VHDL file

6.1.2 Test Bench File

The Test Bench is started in the same manner as usual by establishing the same Inputs and Outputs as the Main VHDL file in the entity.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  -- Uncomment the following library declaration if instantiating
6  entity Clock_5KHz_tb is
7  -- Port ( );
8  end Clock_5KHz_tb;
9  architecture Behavioral of Clock_5KHz_tb is
10 Component Clock_5KHz
11 Port ( CLK100MHZ : in STD_LOGIC;
12       Rst : in STD_LOGIC;
13       FiveKHz : out STD_LOGIC
14 );
15 end Component;
```

Figure 6.2: 5KHz Clock Test Bench File Part 1

Then trigger or stimulus timing is established so that a viable reading can be procured when simulating.

```
17 --Inputs
18 signal CLK100MHZ : STD_LOGIC := '0';
19 signal Rst : STD_LOGIC := '1';
20
21 --Outputs
22 signal FiveKHz : STD_LOGIC;
23
24
25 --Clock definitions
26 constant CLK100MHZ_period : time := 10ns;
27 begin
28 --Instantiate the UUT
29 UUT : Clock_5KHz
30
31 Port Map ( CLK100MHZ => CLK100MHZ,
32           Rst => Rst,
33           FiveKHz => FiveKHz
34 );
35
36 --Clock Process definitions
37 CLK100MHZ_process : process
38 begin
39 CLK100MHZ <= '0';
40 wait for CLK100MHZ_period/2;
41 CLK100MHZ <= '1';
42 wait for CLK100MHZ_period/2;
43 end process;
44 --Stimulus process definitions
45
46 stim_proc: process
47
48 begin
49 wait for 1ms;
50
51 Rst <= '0';
52 wait for 2ms;
53 Rst <= '1';
54
55 wait;
56 end process;
57 end Behavioral;
```

What does this mean?

Figure 6.3: 5KHz Clock Test Bench File Part 2

6.2 MUX DEMUX

As there is no wish to use all eight displays on the board, save for the rightmost three displays, with a Unit, a Ten, and a Hundred digit displayed on each, a Multiplexer/Demultiplexer is required.

6.2.1 Main VHDL File

The entity takes in the Input Ports which tie to the Switches on the Board and has the four control bits for the Seven Segment Display and three bits to control the Anodes of the Displays set as Output Ports.

The Architecture has two internal signals declared, with cnt used to count through loops, and Q to produce an output (AN0 to 2) resulting from the processes in the Behavioural architecture.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  use IEEE.std_logic_unsigned.all;
5  entity Mux_Demux is
6  Port ( FiveKHz : in STD_LOGIC;
7        Rst : in STD_LOGIC;
8        AN0,AN1,AN2 : out STD_LOGIC;
9        Un0,Un1,Un2,Un3 : in STD_LOGIC;
10       T0,T1,T2,T3 : in STD_LOGIC;
11       R0,R1 : in STD_LOGIC;
12       A,B,C,D : out STD_LOGIC
13     );
14 end Mux_Demux;
15 architecture Behavioral of Mux_Demux is
16   signal cnt : std_logic_vector (1 downto 0);
17   signal Q : std_logic_vector (2 downto 0);
18 begin
19   process (FiveKHz, Rst)
20   begin
21     if Rst = '1' then
22       cnt <= "11";
23     elsif rising_edge (FiveKHz) then
24       cnt <= cnt + "01";
25     end if;
26   end process;
27   process (cnt(1), cnt(0))
28   begin
29     if (cnt(1)='0' and cnt(0)='0') then Q <= "110";
30     elsif (cnt(1)='0' and cnt(0)='1') then Q <= "101";
31     elsif (cnt(1)='1' and cnt(0)='0') then Q <= "011";
32     else Q <= "111";
33   end if;
34 end process;
```

Figure 6.4: Mux Demux VHDL File Part 1

The following if and elsif statements process the Switch inputs depending on which of the three displays is to show a digit, and a binary 4-bit signal is produced to be outputted to the Seven Segment Display.

```

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

```

Figure 6.5: Mux Demux VHDL File Part 2

6.2.2 Test Bench File

The entity as before mirrors the Main VHDL file with the same Inputs and Outputs.

```

1 : library IEEE;
2 : use IEEE.STD_LOGIC_1164.ALL;
3 : entity Mux_Demux_tb is
4 :   -- Port ( ); No need to add info here as this is a testbench
5 : end Mux_Demux_tb;
6 : architecture Behavioral of Mux_Demux_tb is
7 :   component Mux_Demux -- Enter file to be tested
8 :     -- Copy the Port information from the VHDL file
9 :     Port ( FiveKHz : in STD_LOGIC;
10 :      Rst : in STD_LOGIC;
11 :      AN0,AN1,AN2 : out STD_LOGIC;
12 :      Un0,Un1,Un2,Un3 : in STD_LOGIC;
13 :      T0,T1,T2,T3 : in STD_LOGIC;
14 :      H0,H1 : in STD_LOGIC;
15 :      A,B,C,D : out STD_LOGIC
16 :    );
17 : end component;

```

Figure 6.6: Mux Demux Test Bench File Part 1

The Input signals are initialised as 0, and the Output signals are established.

```
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 := '0';
23 signal Un1 : std_logic := '1';
24 signal Un2 : std_logic := '0';
25 signal Un3 : std_logic := '0';
26 signal T0 : std_logic := '0';
27 signal T1 : std_logic := '0';
28 signal T2 : std_logic := '1';
29 signal T3 : std_logic := '0';
30 signal H0 : std_logic := '0';
31 signal H1 : std_logic := '1';
32
33 -- output signals
34 -- Example signal Q1 : std_logic := '0';
35
36 signal AN0 : std_logic;
37 signal AN1 : std_logic;
38 signal AN2 : std_logic;
39 signal A : std_logic;
40 signal B : std_logic;
41 signal C : std_logic;
42 signal D : std_logic;
```

Figure 6.7: Mux Demux Test Bench File Part 2

A time constant is set to a period of 8ns to make simulation measurement easier without excessive zooming.

```
46 -- clock period definitions
47 -- Example constant A_period : time := 8ns;
48 constant FiveKHz_period : time := 200us;
49 begin
50 -- Instantiate the UUT
51 UUT : Mux_Demux -- Enter file to be tested
52 Port Map (
53 -- Example A => A,
54 FiveKHz => FiveKHz,
55 Rst => Rst,
56 AN0 => AN0,
57 AN1 => AN1,
58 AN2 => AN2,
59 Un0 => Un0,
60 Un1 => Un1,
61 Un2 => Un2,
62 Un3 => Un3,
63 T0 => T0,
64 T1 => T1,
65 T2 => T2,
66 T3 => T3,
67 H0 => H0,
68 H1 => H1,
69 A => A,
70 B => B,
71 C => C,
72 D => D
73 );
```

Figure 6.8: Mux Demux Test Bench File Part 3

Clock pulses are established with a 50% On and Off duration.

```
75 -- clock process definitions
76 clk_process :process
77 begin
78   FiveKHz <= '0';
79   wait for FiveKHz_period/2; --for 5 ns signal is '0'.
80   FiveKHz <= '1';
81   wait for FiveKHz_period/2; --for next 5 ns signal is '1'.
82 end process;
83
84 -- Stimulus process
85 stim_proc: process
86 begin
87   wait for 1ms;
88   Rst <= '0';
89   wait for 5ms;
90   Rst <= '1';
91   wait;
92 end process;
93 end Behavioral;
```

*Good understanding shown on
Mux/Demux*

6.3 BCD TO SEVEN SEGMENT DISPLAY

6.3.1 Main VHDL File

The 4-bit input from the Mux Demux, and the outgoing Seven Segment Elements are added to the entity.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  entity BCD_7seg is
4    Port ( A,B,C,D : in std_logic;
5           sa,sb,sc,sd,se,sf,sg : out std_logic
6         );
7
8  end BCD_7seg;
```

Figure 6.9: BCD to Seven Segment Display VHDL File Part 1

An internal signal of Q is used, which is set using the incoming signal from the Mux Demux, and the corresponding configuration of the Seven Segment Display is established using the case statement. At the end of the case, there is a 'blank' statement where a binary number of greater than 1001 turns the display off. *Good*

Once the case statement is run through, the segment binary value states are set to Output to the individual segments of the display.

```

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

```

Figure 6.10: BCD to Seven Segment Display VHDL File Part 2

Good understanding shown in design.

6.3.2 Test Bench File

Inputs and outputs for the Test Bench match the Main VHDL design.

```

1  : LIBRARY ieee;
2  : USE ieee.std_logic_1164.ALL;
3  : -- Uncomment the following library declaration if using
4  : -- arithmetic functions with Signed or Unsigned values
5  : --USE ieee.numeric_std.ALL;
6  ENTITY BCD_7seg_tb IS
7  END BCD_7seg_tb;
8  ARCHITECTURE behavior OF BCD_7seg_tb IS
9  : -- Component Declaration for the Unit Under Test (UUT)
10 : COMPONENT BCD_7seg
11 : PORT(
12 : A : IN std_logic;
13 : B : IN std_logic;
14 : C : IN std_logic;
15 : D : IN std_logic;
16 : sa : OUT std_logic;
17 : sb : OUT std_logic;
18 : sc : OUT std_logic;
19 : sd : OUT std_logic;
20 : se : OUT std_logic;
21 : sf : OUT std_logic;
22 : sg : OUT std_logic;
23 : );
24 : END COMPONENT;

```

Figure 6.11: BCD to Seven Segment Display Test Bench File Part 1

Signals are declared and initialised.

```

26 --Inputs
27 signal A : std_logic := '0';
28 signal B : std_logic := '0';
29 signal C : std_logic := '0';
30 signal D : std_logic := '0';
31
32 --Outputs
33 signal sa : std_logic;
34 signal sb : std_logic;
35 signal sc : std_logic;
36 signal sd : std_logic;
37 signal se : std_logic;
38 signal sf : std_logic;
39 signal sg : std_logic;
40
41 -- No clocks detected in port list. Replace <clock> below with
42 -- appropriate port name
43 constant A_period : time := 80 ns;
44 constant B_period : time := 40 ns;
45 constant C_period : time := 20 ns;
46 constant D_period : time := 10 ns;
47 BEGIN
48 -- Instantiate the Unit Under Test (UUT)
49 uut: BCD_7seg PORT MAP (
50   A => A,
51   B => B,
52   C => C,
53   D => D,
54   sa => sa,
55   sb => sb,
56   sc => sc,
57   sd => sd,
58   se => se,
59   sf => sf,
60   sg => sg
61 );
62

```

Figure 6.12: BCD to Seven Segment Display Test Bench File Part 2

The clock process then runs through the binary permutations for the input to the Seven Segment Display.

```

60 -- Clock process definitions
61 A_process :process
62 begin
63   A <= '0';
64   wait for A_period/2;
65   A <= '1';
66   wait for A_period/2;
67 end process;
68 B_process :process
69 begin
70   B <= '0';
71   wait for B_period/2;
72   B <= '1';
73   wait for B_period/2;
74 end process;
75 C_process :process
76 begin
77   C <= '0';
78   wait for C_period/2;
79   C <= '1';
80   wait for C_period/2;
81 end process;
82 D_process :process
83 begin
84   D <= '0';
85   wait for D_period/2;
86   D <= '1';
87   wait for D_period/2;
88 end process;
89 end behavior;

```

Figure 6.13: BCD to Seven Segment Display Test Bench File Part 3

Good understanding shown on test bench design.

6.3.3 Constraints File

The constraints file contains Board references only for the Switches and the Segmented Display as only these elements are required for testing.

```

1  ##Switches
2
3  set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { SW0 }]; #IO_L6N_T3_R0P_16 Schwan[0]
4  set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { SW1 }]; #IO_L5N_T0_D08_BM0CLH_14 Schwan[1]
5  set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { SW2 }]; #IO_L6N_T0_D08_VREF_14 Schwan[2]
6  set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { SW3 }]; #IO_L1N_T1_NRC0_14 Schwan[3]
7  set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { SW4 }]; #IO_L1N_T1_NRC0_14 Schwan[4]
8  set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { SW5 }]; #IO_L7N_T1_D10_14 Schwan[5]
9  set_property -dict { PACKAGE_PIN U10 IOSTANDARD LVCMOS33 } [get_ports { SW6 }]; #IO_L1N_T0_A13_D09_14 Schwan[6]
10 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { SW7 }]; #IO_L6N_T0_D07_14 Schwan[7]
11
12 set_property -dict { PACKAGE_PIN R6 IOSTANDARD LVCMOS33 } [get_ports { D }]; #IO_L54P_T3_35 Schwan[12]
13 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { C }]; #IO_L20P_T3_A09_D04_14 Schwan[13]
14 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { B }]; #IO_L18P_T3_A09_D05_VREF_14 Schwan[14]
15 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { A }]; #IO_L21P_T3_D05_14 Schwan[15]
16
17 ##7 segment display
18
19 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { sa }]; #IO_L24P_T3_A02_D16_14 Schwan
20 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { sb }]; #IO_L5_14 Schwan
21 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { sc }]; #IO_L5_16 Schwan
22 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { sd }]; #IO_L17P_T3_A16_14 Schwan
23 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { se }]; #IO_L13P_T3_NRC0_14 Schwan
24 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { sf }]; #IO_L19P_T3_A10_D06_14 Schwan
25 set_property -dict { PACKAGE_PIN L10 IOSTANDARD LVCMOS33 } [get_ports { sg }]; #IO_L4P_T0_D04_14 Schwan
26
27 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_18 Schwan[0]
28 set_property -dict { PACKAGE_PIN J10 IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L20N_T3_FFE_B_18 Schwan[1]
29 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Schwan[2]
30 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T3_A02_18 Schwan[3]
31 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L6N_T1_D12_14 Schwan[4]
32 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T3_NRC0_14 Schwan[5]
33 set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L24P_T3_35 Schwan[6]
34 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L20N_T3_A02_D18_14 Schwan[7]

```

Figure 6.14: BCD to Seven Segment Display Constraints File

6.4 FINAL BUILD

This final build incorporates the previous three designs with were packaged into IPs of their own, and then imported into the Final Build Project.

6.4.1 Main VHDL File

All the necessary Ports for each IP are included for operation of the design as well as a switch (SW14) which ensures that the unused Seven Segment Displays remain off during operation.

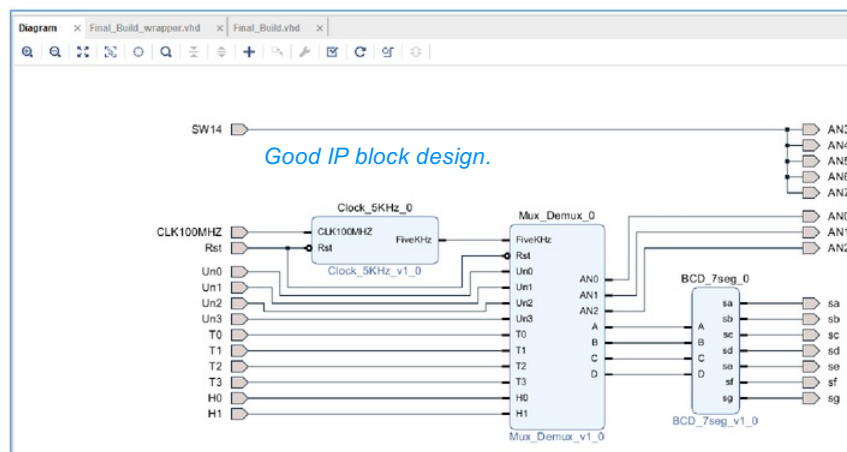


Figure 6.15: Final Build VHDL Block Diagram View

The following VHDL code was generated after importing and linking the previously packaged IPs.

Figure 6.16: Final Build VHDL File Part 1

Figure 6.17: : Final Build VHDL File Part 2

Figure 6.18: : Final Build VHDL File Part 3

```

97 : signal BCD_7seg_0_sa : STD_LOGIC;
98 : signal BCD_7seg_0_sb : STD_LOGIC;
99 : signal BCD_7seg_0_sc : STD_LOGIC;
100 : signal BCD_7seg_0_sd : STD_LOGIC;
101 : signal BCD_7seg_0_se : STD_LOGIC;
102 : signal BCD_7seg_0_sf : STD_LOGIC;
103 : signal BCD_7seg_0_sg : STD_LOGIC;
104 : signal CLK100MHZ_1 : STD_LOGIC;
105 : signal Clock_5KHz_0_FiveKHz : STD_LOGIC;
106 : signal H0_1 : STD_LOGIC;
107 : signal H1_1 : STD_LOGIC;
108 : signal Mux_Demux_0_A : STD_LOGIC;
109 : signal Mux_Demux_0_AN0 : STD_LOGIC;
110 : signal Mux_Demux_0_AN1 : STD_LOGIC;
111 : signal Mux_Demux_0_AN2 : STD_LOGIC;
112 : signal Mux_Demux_0_B : STD_LOGIC;
113 : signal Mux_Demux_0_C : STD_LOGIC;
114 : signal Mux_Demux_0_D : STD_LOGIC;
115 : signal Rst_1 : STD_LOGIC;
116 : signal SW14_1 : STD_LOGIC;
117 : signal T0_1 : STD_LOGIC;
118 : signal T1_1 : STD_LOGIC;
119 : signal T2_1 : STD_LOGIC;
120 : signal T3_1 : STD_LOGIC;
121 : signal Un0_1 : STD_LOGIC;
122 : signal Un1_1 : STD_LOGIC;
123 : signal Un2_1 : STD_LOGIC;
124 : signal Un3_1 : STD_LOGIC;

```

Figure 6.19: : Final Build VHDL File Part 4

```

125 : begin
126 : AN0 <= Mux_Demux_0_AN0;
127 : AN1 <= Mux_Demux_0_AN1;
128 : AN2 <= Mux_Demux_0_AN2;
129 : AN3 <= SW14_1;
130 : AN4 <= SW14_1;
131 : AN5 <= SW14_1;
132 : AN6 <= SW14_1;
133 : AN7 <= SW14_1;
134 : CLK100MHZ_1 <= CLK100MHZ;
135 : H0_1 <= H0;
136 : H1_1 <= H1;
137 : Rst_1 <= Rst;
138 : SW14_1 <= SW14;
139 : T0_1 <= T0;
140 : T1_1 <= T1;
141 : T2_1 <= T2;
142 : T3_1 <= T3;
143 : Un0_1 <= Un0;
144 : Un1_1 <= Un1;
145 : Un2_1 <= Un2;
146 : Un3_1 <= Un3;
147 : sa <= BCD_7seg_0_sa;
148 : sb <= BCD_7seg_0_sb;
149 : sc <= BCD_7seg_0_sc;
150 : sd <= BCD_7seg_0_sd;
151 : se <= BCD_7seg_0_se;
152 : sf <= BCD_7seg_0_sf;
153 : sg <= BCD_7seg_0_sg;
154 : BCD_7seg_0: component Final_Build_BCD_7seg_0_0
155 : port map (
156 : A => Mux_Demux_0_A,
157 : B => Mux_Demux_0_B,
158 : C => Mux_Demux_0_C,
159 : D => Mux_Demux_0_D,
160 : sa => BCD_7seg_0_sa,
161 : sb => BCD_7seg_0_sb,
162 : sc => BCD_7seg_0_sc,
163 : sd => BCD_7seg_0_sd,
164 : se => BCD_7seg_0_se,
165 : sf => BCD_7seg_0_sf,
166 : sg => BCD_7seg_0_sg
167 : );

```

Figure 6.20: Final Build VHDL File Part 5

No explanation given!

```

168 Clock_5MHz_0: component Final_Build_Clock_5MHz_0_0
169 port map (
170   CLK100MHz => CLK100MHz_1,
171   FiveKHz => Clock_5MHz_0_FiveKHz,
172   Rst => Rst_1
173 );
174 Mux_Demux_0: component Final_Build_Mux_Demux_0_0
175 port map (
176   A => Mux_Demux_0_A,
177   AN0 => Mux_Demux_0_AN0,
178   AN1 => Mux_Demux_0_AN1,
179   AN2 => Mux_Demux_0_AN2,
180   B => Mux_Demux_0_B,
181   C => Mux_Demux_0_C,
182   D => Mux_Demux_0_D,
183   FiveKHz => Clock_5MHz_0_FiveKHz,
184   H0 => H0_1,
185   H1 => H1_1,
186   Rst => Rst_1,
187   T0 => T0_1,
188   T1 => T1_1,
189   T2 => T2_1,
190   T3 => T3_1,
191   Un0 => Un0_1,
192   Un1 => Un1_1,
193   Un2 => Un2_1,
194   Un3 => Un3_1
195 );
196 end STRUCTURE;

```

Figure 6.21: Final Build VHDL File Part 6

6.4.2 Constraints File

The constraints file has the required Port references that are needed by the three IPs brought into the Final Build.

```

1: ## Clock signal
2: set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports { CLK100MHz }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
3: create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHz}];
4:
5: ##Switches
6:
7: set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { Un0 }]; #IO_L24H_T3_R00_15 Sch=sv[0]
8: set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { Un1 }]; #IO_L3N_T0_DQS_EN0CLK_14 Sch=sv[1]
9: set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { Un2 }]; #IO_L4H_T0_DQS_VREF_14 Sch=sv[2]
10: set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { Un3 }]; #IO_L13N_T1_MRCC_14 Sch=sv[3]
11: set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { T0 }]; #IO_L12N_T1_MRCC_14 Sch=sv[4]
12: set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { T1 }]; #IO_L7N_T1_D10_14 Sch=sv[5]
13: set_property -dict { PACKAGE_PIN U10 IOSTANDARD LVCMOS33 } [get_ports { T2 }]; #IO_L17N_T2_A13_D09_14 Sch=sv[6]
14: set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { T3 }]; #IO_L8N_T0_D07_14 Sch=sv[7]
15: set_property -dict { PACKAGE_PIN T0 IOSTANDARD LVCMOS18 } [get_ports { H0 }]; #IO_L24H_T3_14 Sch=sv[8]
16: set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { H1 }]; #IO_25_34 Sch=sv[9]
17: #set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { SW10 }]; #IO_L18P_T3_D00_R00F_B_14 Sch=sv[10]
18: #set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { SW11 }]; #IO_L33P_T3_A03_D19_14 Sch=sv[11]
19: #set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { SW12 }]; #IO_L24P_T3_35 Sch=sv[12]
20: #set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { SW13 }]; #IO_L20P_T3_A04_D04_14 Sch=sv[13]
21: set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { SW14 }]; #IO_L18N_T1_A08_D05_VREF_14 Sch=sv[14]
22: set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { Rst }]; #IO_L21P_T3_D08_14 Sch=sv[15]
23:
24: ##7 segment display
25:
26: set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { a0 }]; #IO_L24H_T3_A00_D16_14 Sch=sv[16]
27: set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { a1 }]; #IO_25_14 Sch=sv[17]
28: set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 } [get_ports { a2 }]; #IO_25_15 Sch=sv[18]
29: set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { a3 }]; #IO_L17P_T2_A16_18 Sch=sv[19]
30: set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { a4 }]; #IO_L13P_T2_MRCC_14 Sch=sv[20]
31: set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { a5 }]; #IO_L19P_T3_A10_D06_14 Sch=sv[21]
32: set_property -dict { PACKAGE_PIN L10 IOSTANDARD LVCMOS33 } [get_ports { a6 }]; #IO_L4P_T0_D04_14 Sch=sv[22]
33:
34: #set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L18N_T3_A01_VREF_16 Sch=sv[23]
35:
36: set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { AN0 }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
37: set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { AN1 }]; #IO_L23N_T3_F0E_B_15 Sch=an[1]
38: set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { AN2 }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
39: set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { AN3 }]; #IO_L19P_T3_A22_18 Sch=an[3]
40: set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { AN4 }]; #IO_L8N_T1_D12_14 Sch=an[4]
41: set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { AN5 }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
42: set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { AN6 }]; #IO_L23P_T3_35 Sch=an[6]
43: set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { AN7 }]; #IO_L23N_T3_A00_D18_14 Sch=an[7]

```

Good

Figure 6.22: Final Build VHDL Constraints File

7 ANALYSIS OF XILINX VIVADO SIMULATED RESULTS

7.1 5KHz CLOCK RESULTS AND ANALYSIS

The result recorded, shows the 100MHz Clock reduced to a 5KHz Clock pulse with a cycle of approximately 1000us measured. *Period of 5KHz clock is 200uS!*

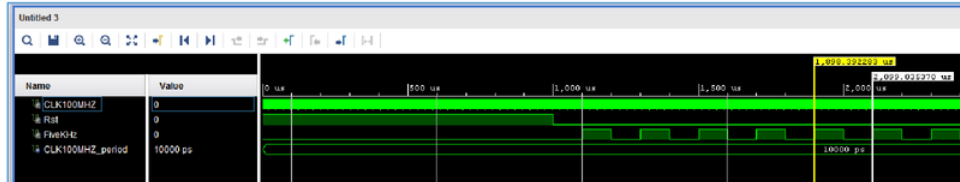


Figure 7.1: Measured at 4ms

The results here we favourable to continue development.

7.2 SEVEN SEGMENT DISPLAY RESULTS AND ANALYSIS

A result was recorded that showed the displaying of '0' by turning only the 'g' element off.

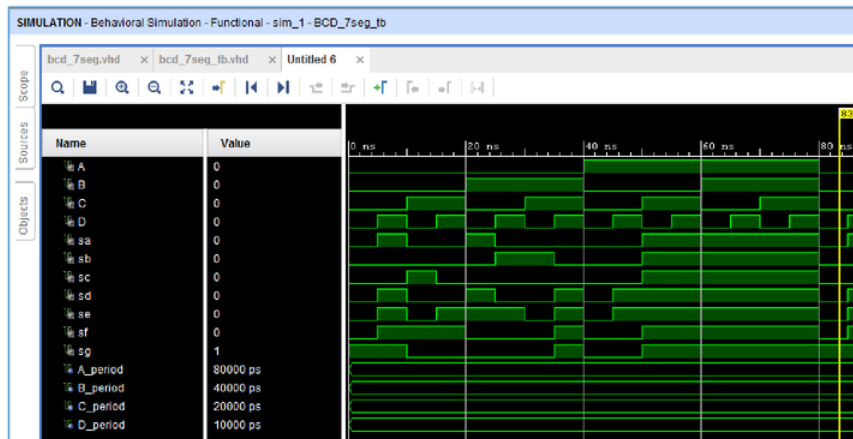
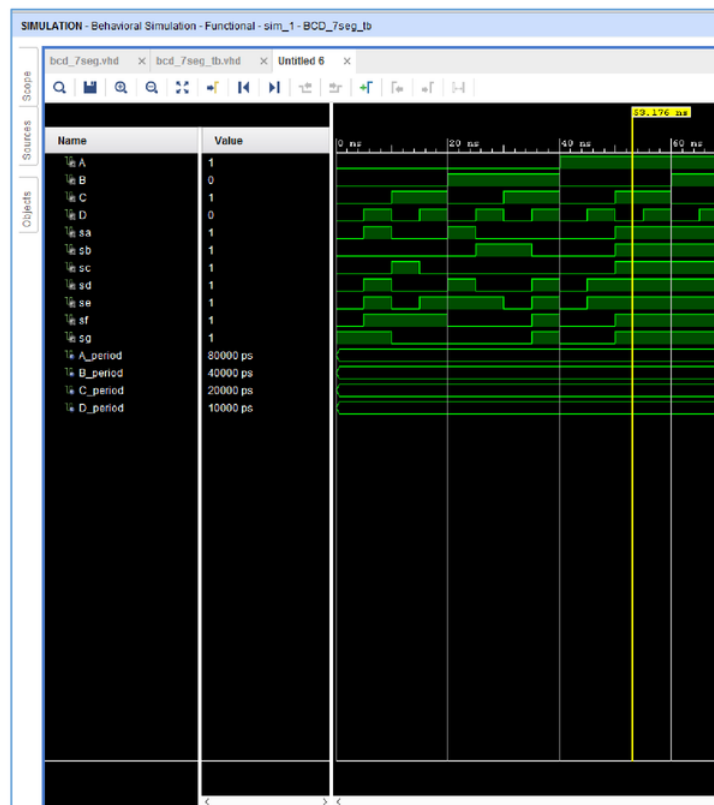


Figure 7.2: Proof of '0' where sg is turned off/1

A binary number of 1010 which is outside the tolerable range has resulted in all segments turning off, rather than show an incorrect digit, as a result of an unexpected error. *Good*



What is this result?

Figure 7.3: Proof that Display turns off when Binary value greater than 1001 is sent

The desired results were achieved in this simulation, and development could proceed to the next step towards the Final Build.

Where is the analysis of the Mux/Demux?

8 ANALYSIS OF NEXYS 4 DDR FPGA PRACTICAL RESULTS

The below photos show the board set to show the numeric values of 12, 42, and 129 on the FPGA board through the setting of switches. The switches were set using the Hundreds, Tens and Units method of digit entry. As the correct values are shown, it can only be surmised that the test was a success.

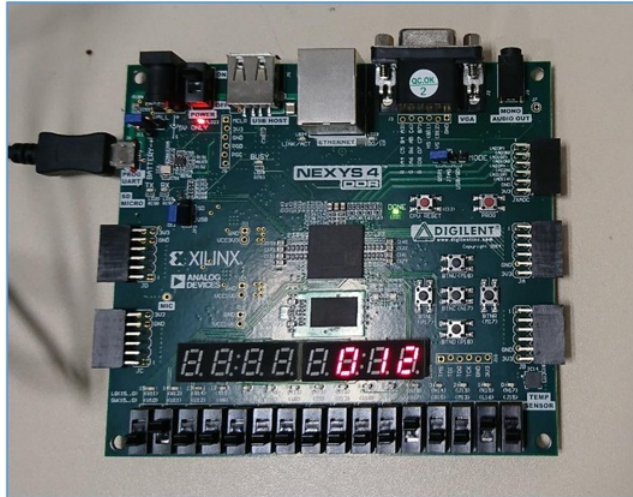


Figure 8.1: Nexys 4 FPGA displaying 12

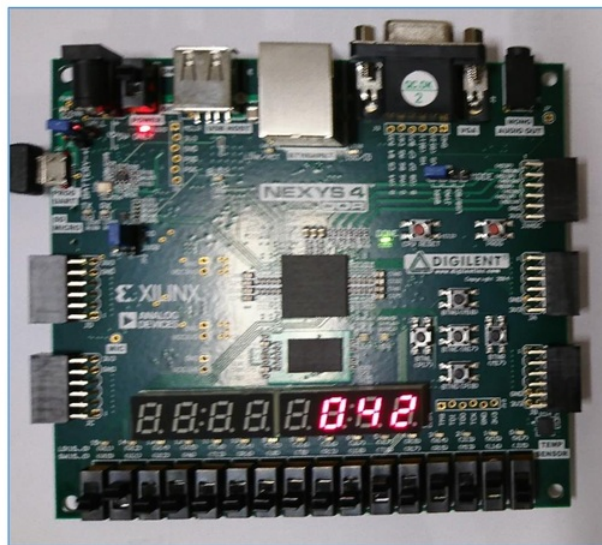


Figure 8.2: Nexys 4 FPGA displaying 42

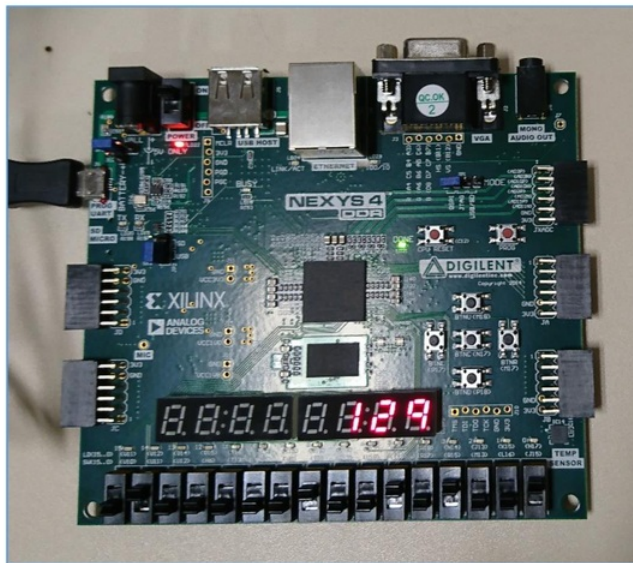


Figure 8.3: Nexys 4 FPGA displaying 129

Good use made of pictorial evidence in showing practical results.

9 CONCLUSION

Through the development of initially the Binary to BCD converter VHDL file, and then further into the BCD to 7 Segment Display VHDL file, the value of modular coding was shown. It was learnt that by creating these individually programmed VHDL files and wrapping them together to create another IP, that in turn can be linked to another IP, from a library of completed IPs to create a larger project, that this method of coding has much in common with other common forms of programming where projects can be built from previously made and tested code that is proven to work. By building a project from previously tested code also provides the benefit of having an easier time of troubleshooting the code due to trustworthiness of previous code. Evidence of testing should always be retained to better facilitate the use of trustworthy code in clear documentation that also records changes made to the code over its lifespan.

The way that the VHDL programming language executes lends to a certain level of confidence in the language as the code will run one line at a time consecutively, which in some ways makes it easier to understand than other programming languages and good basis to perhaps learn how to code.

Satisfactory general conclusion noted which does show some reflection.

10 WORKS CITED

Mealy, B., 2017. *Free Range VHDL*. Barcelona: Free Range Factory.

NationalSemiconductor, 2014. (*Datasheet*) *DM74185A PDF - National Semiconductor : (DM74184 / DM74185A) BCD-to-Binary and Binary-to-BCD Converters (7-page)*. [Online]

Available at: <http://www.datasheetpdf.com/pdf/516096/NationalSemiconductor/DM74185A/7>
[Accessed 05 01 2018].

Readler, B., 2014. *Vhdl By Example*. Washington: Full Arc Press.

Very little research shown.

11 IMAGE SOURCES

- Figure 2.4 <http://www.datasheetspdf.com/pdf/516096/NationalSemiconductor/DM74185A/7>
Figure 2.5 <http://www.datasheetspdf.com/pdf/516096/NationalSemiconductor/DM74185A/7>
Figure 2.6 <https://xlrx.i.lithium.com/t5/image/serverpage/image-id/21855i9AE023EFC8016D1E?v=1.0>

PSDE5029-Assignment_1-Emyr_Wyn_Dyer

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

50/100

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

Text Comment. Very brief Introduction... What do the following terms mean: VHDL; FPGA?

PAGE 6

PAGE 7

Text Comment. Good detail given.

PAGE 8

PAGE 9

Text Comment. Very good detail.

PAGE 10

Text Comment. This diagram may be typical but has no relevance to you showing understanding of how IP blocks are used

PAGE 11

Text Comment. Very good

Text Comment. Good

PAGE 12

PAGE 13

Text Comment. Good explanation given.

PAGE 14

PAGE 15

PAGE 16

Text Comment. Good

PAGE 17

PAGE 18

PAGE 19

PAGE 20

Text Comment. Excellent detail shown in the design and simulation of the 8 bit device

PAGE 21

PAGE 22

Text Comment. Bit more detail on analysis please

Text Comment. Use the simulated results to show your understanding of how the device works!

PAGE 23

PAGE 24

PAGE 25

Text Comment. Good use made of pictorial evidence.

PAGE 26

Text Comment. Design not explained well.

PAGE 27

Text Comment. What does this mean?

PAGE 28

PAGE 29

PAGE 30

Text Comment. Good understanding shown on Mux/Demux.

Text Comment. Good

Text Comment. Good understanding shown in design.

Text Comment. Good understanding shown on test bench design.

Text Comment. Good IP block design.

Text Comment. No explanation given!

Text Comment. Good

Text Comment. Period of 5KHz clock is 200uS!

Text Comment. Good

Text Comment. What is this result?

Text Comment. Where is the analysis of the Mux/Demux?

Text Comment. Good use made of pictorial evidence in showing practical results.

Text Comment. Satisfactory general conclusion noted which does show some reflection.

PAGE 43

Text Comment. Very little research shown.

PAGE 44

GRADING FORM: CSG-RUBIC

EMYR DYER

50

RESEARCH-INFORMED LITERATURE



Very brief Introduction... What do the following terms mean: VHDL; FPGA? Structure of a typical VHDL file: Good detail shown. (3/5) Binary to BCD conversion: Very good detail shown. (4/5) IP Block Design in Vivado: This diagram may be typical but has no relevance to you showing understanding of how IP blocks are used (2/5) Total (9/15)

9

KNOWLEDGE AND UNDERSTANDING



Design of 6/8 bit Binary to BCD Converter: Good 6 bit design shown in report. (3/5); Excellent detail shown in the design and simulation of the 8 bit device (4/5). Re-design to suit new applications: 5KHz clock design not explained well. (1/3); Good understanding shown on Mux/Demux (2/3); 7 segment decoder design shows clear understanding (2/3); Full block design for the adaptation shown(3/6) Total (15/25)

15

ANALYSIS



Analysis of Xilinx Vivado simulation results: 6 bit design: bit more detail on analysis please (2/5); 8 bit design: Use the simulated results to show your understanding of how the device works! (2/5).
Analysis of adapted application results: Very poor set of results obtained. 5KHz clock (1/5) incorrect analysis; Mux/Demux (0/5) no result shown, and 7 segment decoder not explained well(2/5) Total (7/25)

7

PRACTICAL APPLICATION AND DEP.



Downloading and testing to specification on Nexys 4 DDR FPGA board: Good use made of pictorial evidence in analysis of results. (6/10) Downloading and testing of new application results on the Nexys 4 DDR FPGA board: Good use made of pictorial evidence in the correct analysis of results obtained. (6/10) Total (12/20)

12

SKILLS FOR PROFESSIONAL PRAC.



Deployment of appropriate media: good use made of screen dumps and pictorial evidence in design and analysis of results. (3/5) Report layout and readability: 6/8 bit design well presented but adapted design and analysis not clear. (2/5) Appropriate conclusion: Satisfactory general conclusion noted which does show some reflection. (2/5) Total (7/15)

7