

assignment 2

by Liesel Williams

FILE	8808_LIESEL_WILLIAMS_ASSIGNMENT_2_152853_1931029504.DOCX		
TIME SUBMITTED	28-MAR-2018 03:18PM (UTC+0100)	WORD COUNT	4968
SUBMISSION ID	85732707	CHARACTER COUNT	35055



PSDE 6015

Electronic Design

Lecturer – M Hall

Year - 2017-2018

Assignment 2

VENDING MACHINE

Liesel Williams

Contents

1. Introduction	4
2. Research and Background Theory	5
2.1 General design	5
3. Design method, choices and calculations	7
4. Practical Testing and Improvement	9
4.1 Multisim	9
4.2 Multisim PLD	13
4.3 VHDL Design in Vivadoo	24
4.4 Adaptation in VHDL	27
5 Conclusion.....	30
6 References	31
Appendix A	32
Appendix B	41

TABLE OF FIGURES

Figure 1 - Block diagram of vending machine	4
Figure 2 - Adaption of outputs from Fig 1.....	4
Figure 3 - Bottom Up Approach to Design VM	5
Figure 4 - Top Down Approach to Design VM.....	5
Figure 5 - VHDL program structure	6
Figure 6 - Hardware board functionality.....	6
Figure 7 - FSM	7
Figure 8 - Multisim test 1	9
Figure 9 - Multisim test 2	10
Figure 10 - Multisim test 3	11
Figure 11 - Multisim logic analyser	12
Figure 12 - PLD test 1	13
Figure 13 - Nexys test 1 * 7 seg.....	13
Figure 14 - Common anode 7 seg	14
Figure 15 - Display outputs	14
Figure 16 - Clocking the 7 segs.....	14
Figure 17 - Segment logic.....	14
Figure 18 - MUX block.....	15
Figure 19 - MUX TENS	15
Figure 20 - MUX UNITS.....	16
Figure 21 - Clocked MUX TENS.....	17
Figure 22 - MUX logic levels	18
Figure 23 - MUX subcircuit logic	19
Figure 24 - Add third segment P	19
Figure 25 - PLD test 2	20
Figure 26 - Clocking sub-circuit.....	20
Figure 27 - PLD test 3	21
Figure 28 - PLD test 4	22
Figure 29 - Board on test.....	23
Figure 30 - Entity for 70	24
Figure 31 - VM1 Entity	24
Figure 33 - VM1 Architecture.....	24
Figure 34 - Example code for 70	24
Figure 35 - Simulation test 1	25
Figure 36 - Simulation test 2	25
Figure 37 - Simulation test 3	26
Figure 38 – VM2 Entity.....	27
Figure 39 – VM2 Architecture.....	27
Figure 40 - Example code for 90	27
Figure 41 - Simulation test 4	28
Figure 42 - Simulation test 5	28
Figure 43 - Simulation test 6	29

1. Introduction

Demonstration and understanding advanced design methods and modern implementation technical issues within digital electronic design is the main aim of the report.

By using theory and practical testing to analyse the requirements and design an advanced digital electronic system i.e. a vending machine to meet customer specification.

A detailed design specification was acquired and a Finite State machine (FSM) was designed using the 'Moore' model to count 50p, 20p and 10p coins fed into a vending machine up to a value of 70p and output the total value on two seven segment displays.

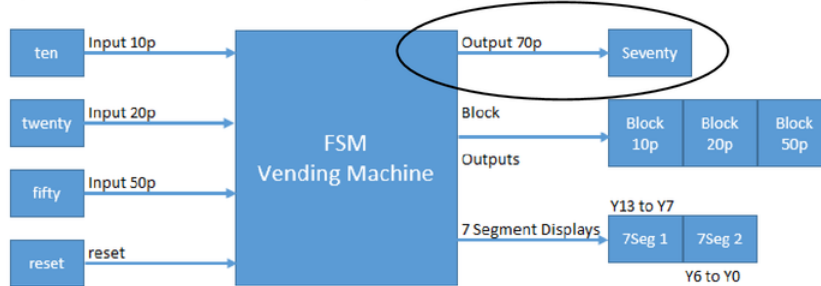


Figure 1 - Block diagram of vending machine

The VHSIC Hardware Descriptive Language (VHDL) code for the design must be adapted to activate specific mechanisms 'Mech 1' and 'Mech 2' upon receipt of either 70p or 90p respectively. For simplification purposes only correct money can be entered and no change is given.

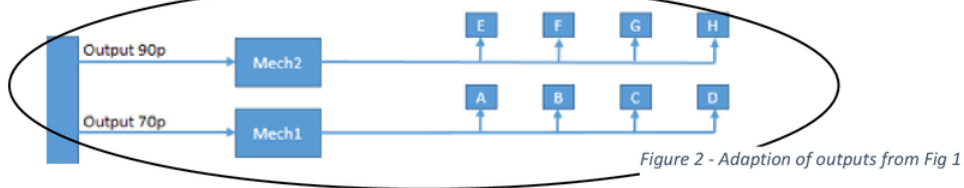


Figure 2 - Adaption of outputs from Fig 1

The completed design must be downloaded to the Digilent Nexys 4 FPGA board, tested against the specification using bespoke test procedures and the results evaluated critically.

SPECIFICATION

The vending machine had the following specification as agreed with the customer:

- Block 10p, 20p and 50p inputs if 70p would be exceeded.
- Only one coin can be input at any one time physically.
- Pressing reset at any point would return the FSM to the initial state of zero coins entered.

2. Research and Background Theory

2.1 General design

The digital design procedure will influence modelling and simulation actions i.e. time available to build models/simulate, complex building blocks being broken down to smaller, simpler parts etc.

The first step in design is to consider the organisation of the project i.e. Top-Down, which gives more precise structured project control or Bottom-Up, which is more experimental in nature.

The system design :-

MULTISIM

Multisim allows the designer a library of pre-made components to choose from e.g. for the vending machine a decoder, multiplexer (MUX) and clock were all available. The Finite State Machine (FSM) was designed from the given specification. This is a bottom up approach.

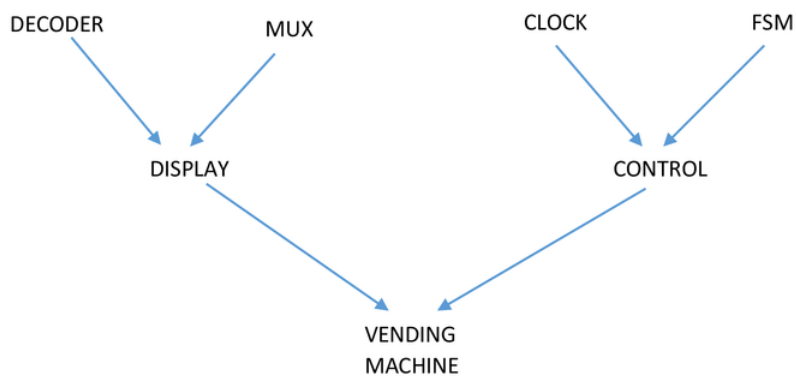


Figure 3 - Bottom Up Approach to Design VM

VHDL

VHDL allows a top down approach, all components are made bespoke to the overall design i.e. written in code so they can be designed in any particular way required. This approach starts with the specification and then designs the required components appropriately.

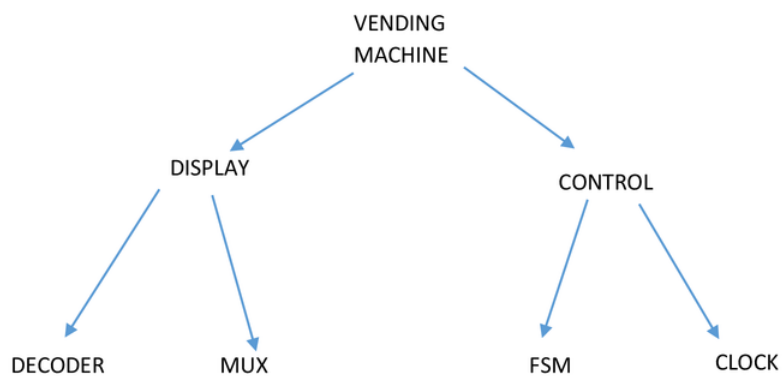


Figure 4 - Top Down Approach to Design VM

VHDL is the VHSIC Hardware Description Language where VHSIC stands for Very High Speed Integrated Circuit. "This programming language describes a logic circuit by function, data flow behaviour and structure, it is used to configure a Programmable Logic Device (PLD) or example a Field Programmable Gate Array (FPGA) with a custom logic design". (Srmuniv.ac.in,2018)

According to (Blaine, 2014) "VHDL code completely describes how the logic design works. And that together with a device-specific file including the implementation details such as device package type, I/O pin assignments etc. the VHDL code is all that's needed to create an operating FPGA" (Readler, 2014)

The basic building units in a VHDL design called Blocks form the logic circuit design. The design starts with an entity block describing the interfaces. Next, the architecture block describes the internal operation and there are also several functional blocks to build up the design elements in the logic circuit. Once built the design can be simulated and synthesized to test the logic operations. Simulation just tests the basic logic against the concept of the design. Synthesis includes timing and outside influences of the FPGA to effect the test and is therefore a more thorough to carry out.

The following image shows the VHDL program structure.

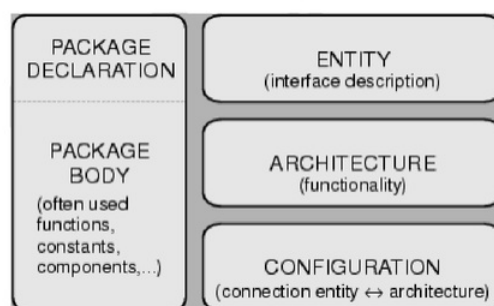
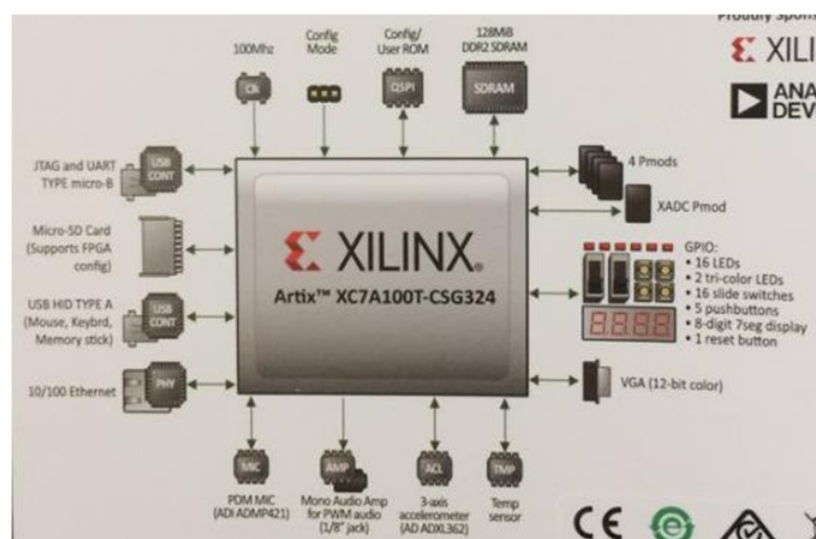


Figure 5 - VHDL program structure

This image shows the various options available for the NEXYS 4 Artix 7 FPGA board that was used to test the vending machine design.



(Reference.digilentinc.com, 2018)

Figure 6 - Hardware board functionality

3. Design method, choices and calculations

Working from the specification of the vending machine, the two design methods adopted for comparison purposes are e-CAD (multisim PLD) and Vivadoo (VHDL). These methods were used to design the logic of the machine with switches used for inputs and an LED as the output state indicator.

A finite state machine was developed as shown below.

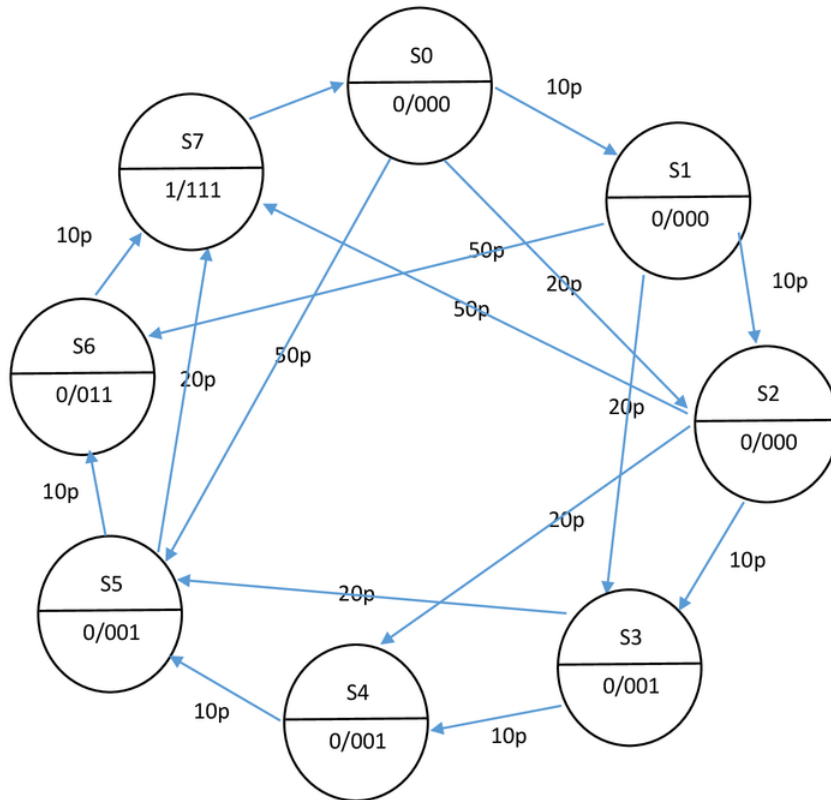


Figure 7 - FSM

The inputs are 10p, 20p, 50p and reset. The outputs are Mech 70 for the correct operation of a mechanism to release the product purchased. For the adaptation, another output called Mech 90 is required.

Design steps:

- State allocation was chosen for the FSM design shown above i.e. S0 = reset, S1 = 10p total, S2 = 20p total etc.
- Following the state diagram with all possible operation modes considered a state table was drawn up.
- Karnaugh mapping used to reduce the Boolean expressions to a minimum.

- On completion of the logic design the circuit was built and tested in multisim.
- PLD multisim was then used to provide a downloadable version for the Nexys 4 board.
- This version was tested using one display only to check the logic (0 and P displays do not change)
- A multiplexing system was designed to allow two displays to run simultaneously.
- A Vivadoo project was created and the VHDL code written to achieve the same results.
- Testing was carried out using a logic analyser initially.
- Final testing on the Nexys 4 board was undertaken.
- Adaptation for the 90p input took place on the VHDL code.

From the state diagram this truth table was constructed

CURRENT		NEXT				0p	10p	20p	50p	OUTPUT			
		0p	10p	20p	50p	JF=00	JF=01	JF=10	JF=11				
Q _A Q _B Q _C		JF=00	JF=01	JF=10	JF=11	Q _A ⁺ Q _B ⁺ Q _C ⁺	Q _A ⁺ Q _B ⁺ Q _C ⁺	Q _A ⁺ Q _B ⁺ Q _C ⁺	Q _A ⁺ Q _B ⁺ Q _C ⁺	70p	B10	B20	B50
S0	000	S0	S1	S2	S5	000	001	010	101	0	0	0	0
S1	001	S1	S2	S3	S6	001	010	011	110	0	0	0	0
S2	010	S2	S3	S4	S7	010	011	100	111	0	0	0	0
S3	011	S3	S4	S5	S7	011	100	101	111	0	0	0	1
S4	100	S4	S5	S6	S7	100	101	110	111	0	0	0	1
S5	101	S5	S6	S7	S7	101	110	111	111	0	0	0	1
S6	110	S6	S7	S7	S7	110	111	111	111	0	0	1	1
S7	111	S7	S7	S7	S7	111	111	111	111	1	1	1	1

As only one coin can be physically entered at any one time then the inputs reduce by using x for 'don't care'

10	20	50	J	F
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

This reduces to 2 flip flops for 4 states and the reduced expressions from the K maps are:

$$Qa^+ = A + JB + FJ + FBC$$

$$Qb^+ = JA + AB + NJNFB + FNBC + FBNC + JNFNB + JFC$$

$$Qc^+ = NFC + FNC + JAC + JFB + FAB$$

Now the logic circuitry is produced.

4. Practical Testing and Improvement

4.1 Multisim

This section documents the tests of the design on multisim at each stage, the four inputs have been set as push button switches 1,2,3 and spacebar.

Three displays were used of which the second and third were hard wired to show 0 and P, the first displayed the result of the logic. This design was thoroughly tested as shown below.

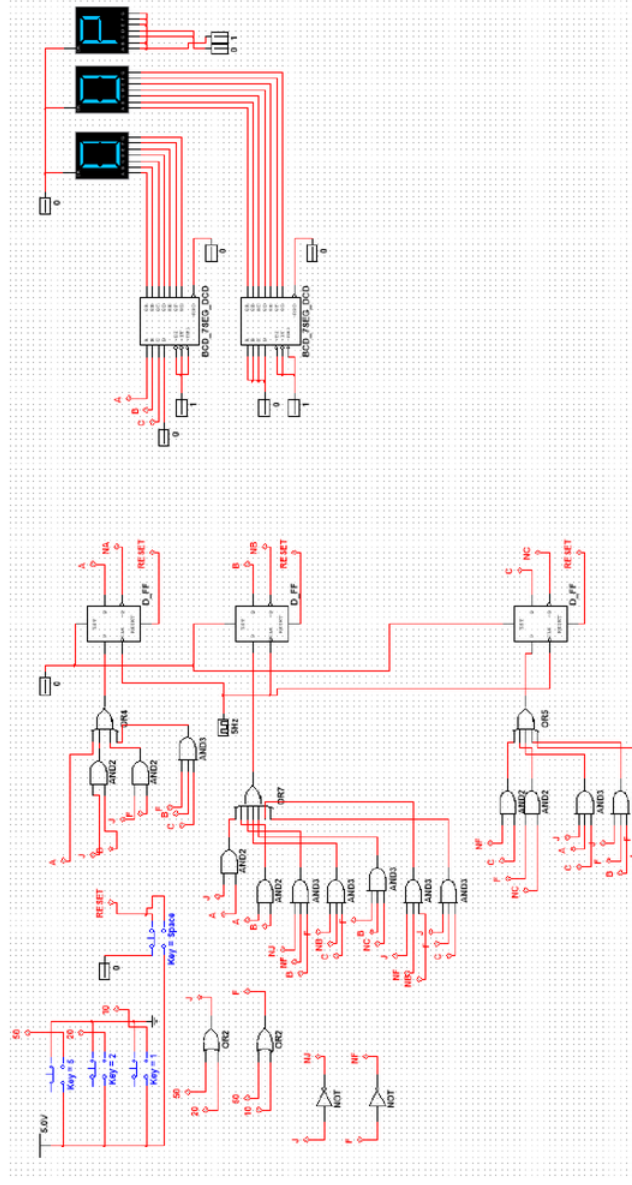


Figure 8 - Multisim test 1

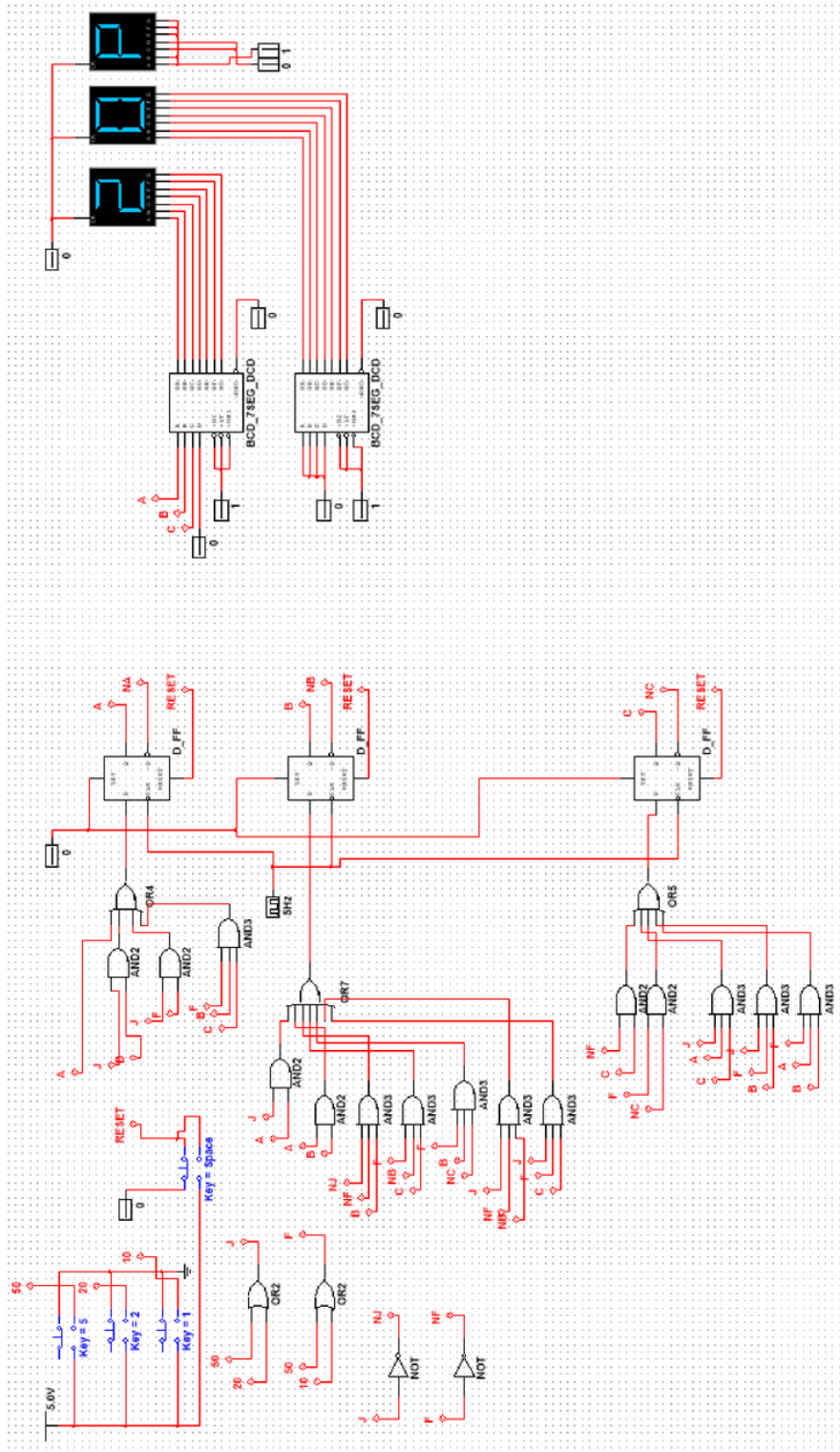


Figure 9 - Multisim test 2

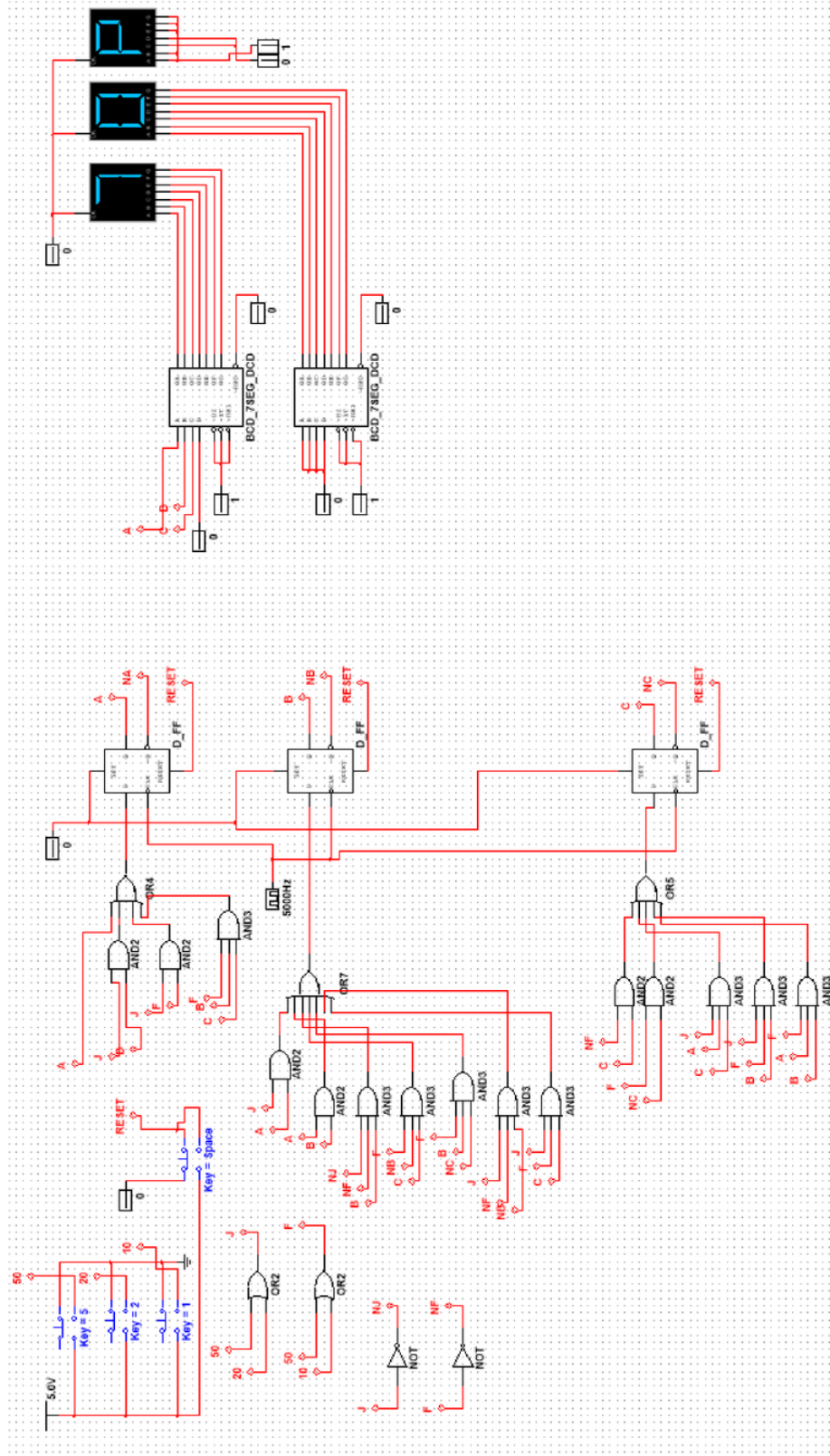


Figure 10 - Multisim test 3

Another method of testing available on the multisim is the use of a logic analyser, this can be used to probe any of the lines at input/output and discover what each logic level is.

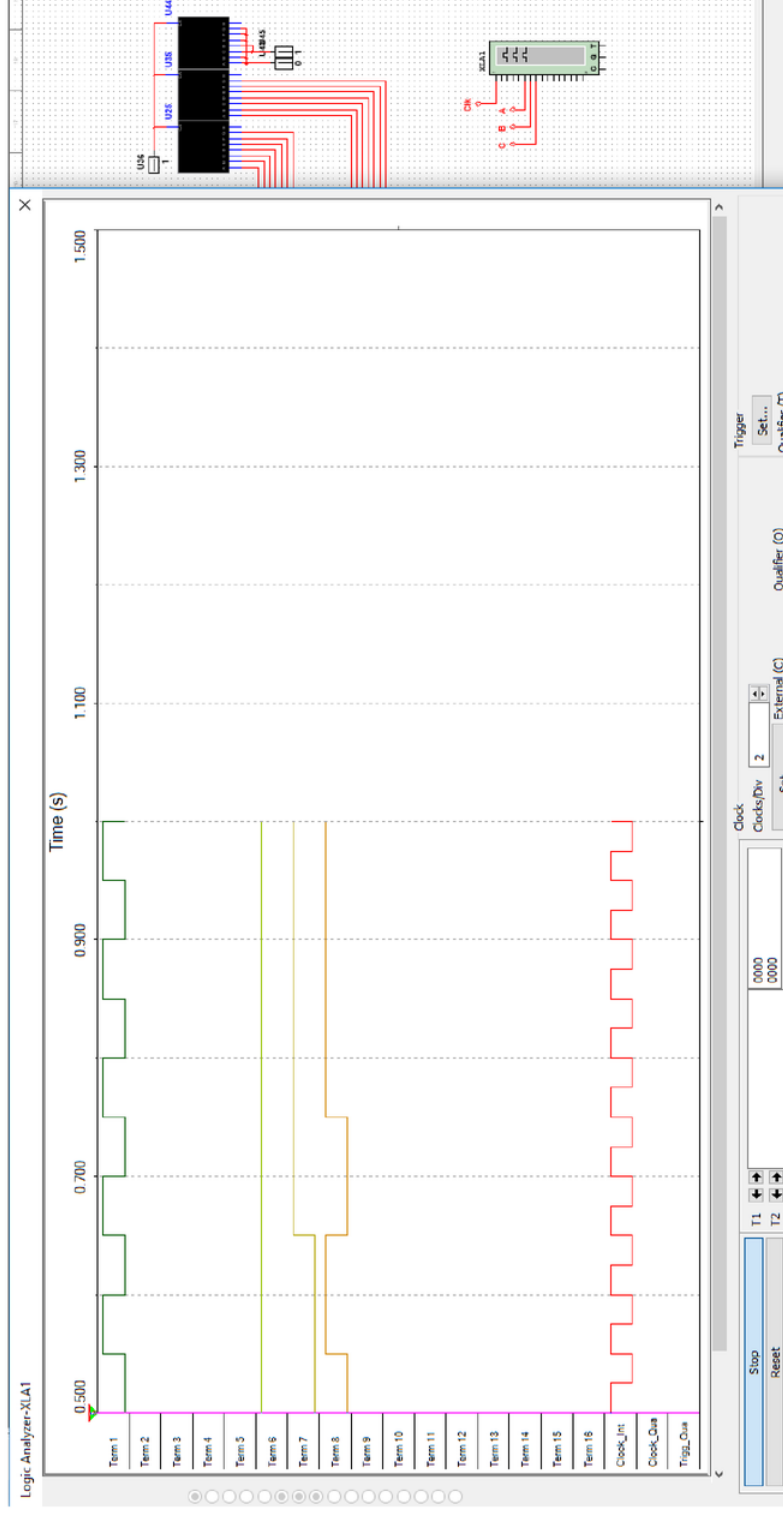


Figure 11 - Multisim logic analyser

4.2 Multisim PLD

Once satisfied with the design a PLD version was drawn up as seen below.

Design on PLD multisim with initially only the tens seven segment display working.

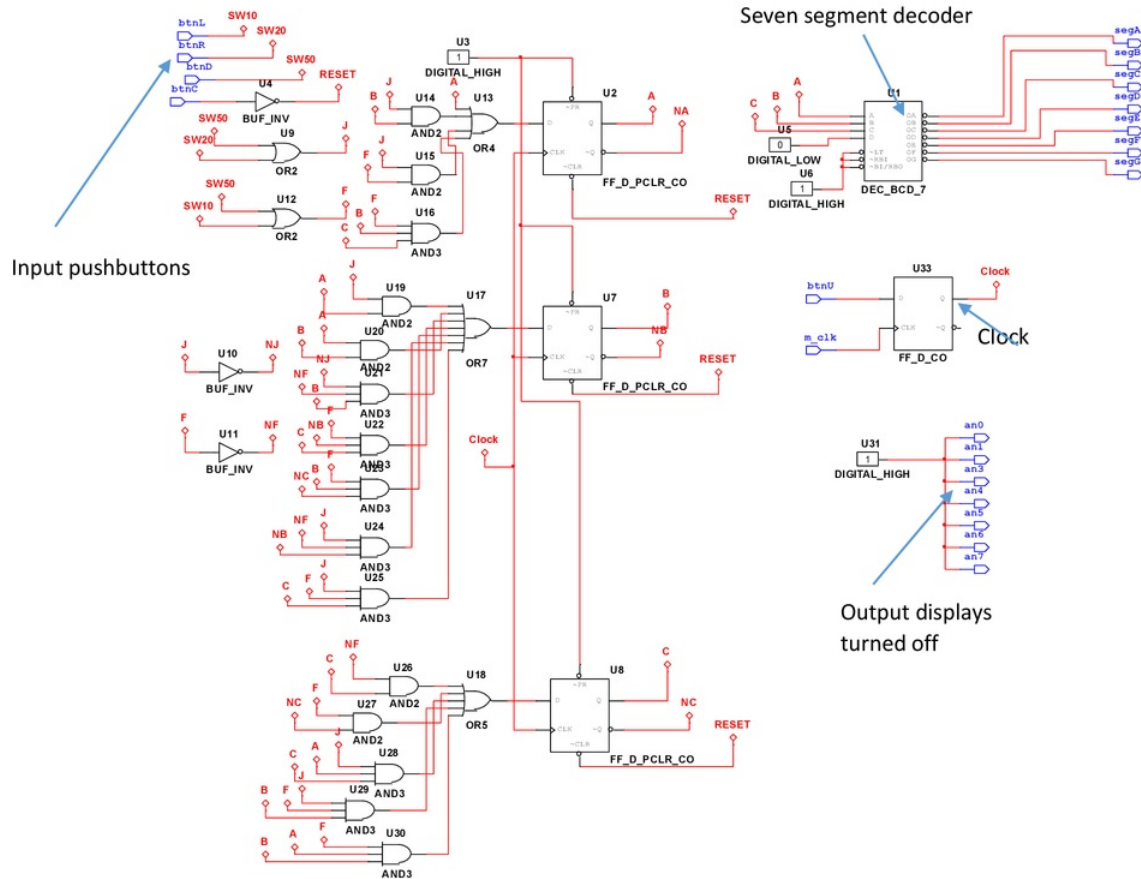


Figure 12 - PLD test 1

The design was tested for the TENS display only.

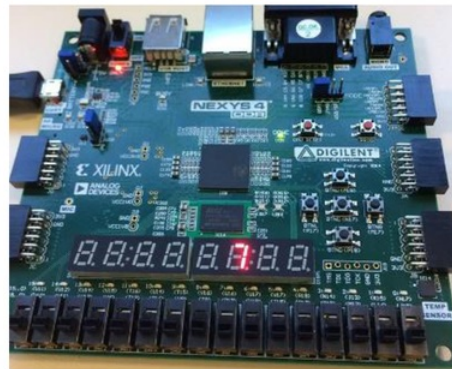


Figure 13 - Nexys test 1 * 7 seg

The next step is to design and build a multiplexer to get more than one display working.

Three methods were considered namely Three-state logic, multiplexers and Boolean.

Three-state logic or tri-state logic has a third state (high-impedance state) added effectively removing the port from the circuit, this utilises complex circuitry to switch between enabling **an0** and **an1** for the logic code for UNITS/ TENS and interfaced with OR gates so that either 0 or 1 will activate.

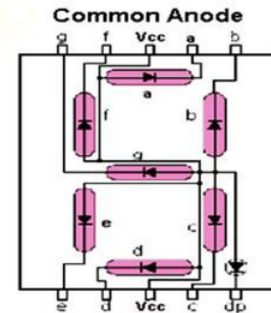
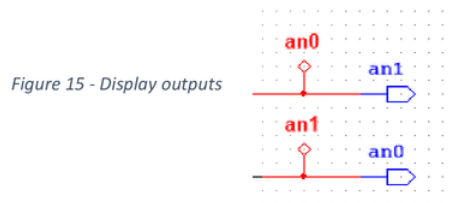


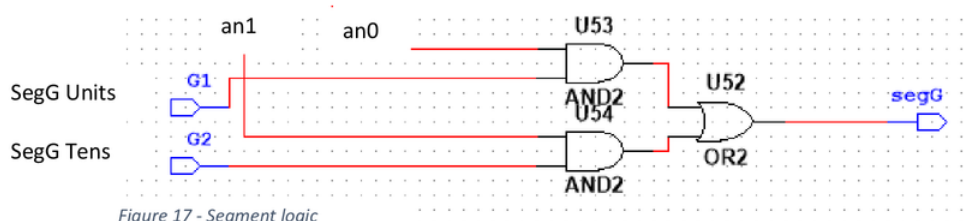
Figure 14 - Common anode 7 seg

The seven segment displays used are common anode, meaning there is a digital 0 on cathode and applying a digital 0 will light up each segment, as shown in the schematic below.

The other method is logic gates constructed to give the correct BCD for the display.



This simple arrangement allows an0 and an1 to alternate back and forth on the clock pulse.



Above segment g is controlled via two AND gates and an OR gate, similarly all segments are controlled.

The third method is the MUX block available for multiplexing.

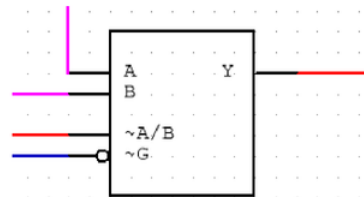


Figure 18 - MUX block

Design process of multiplexer for alternating between TENS and UNITS, tested using indicators.

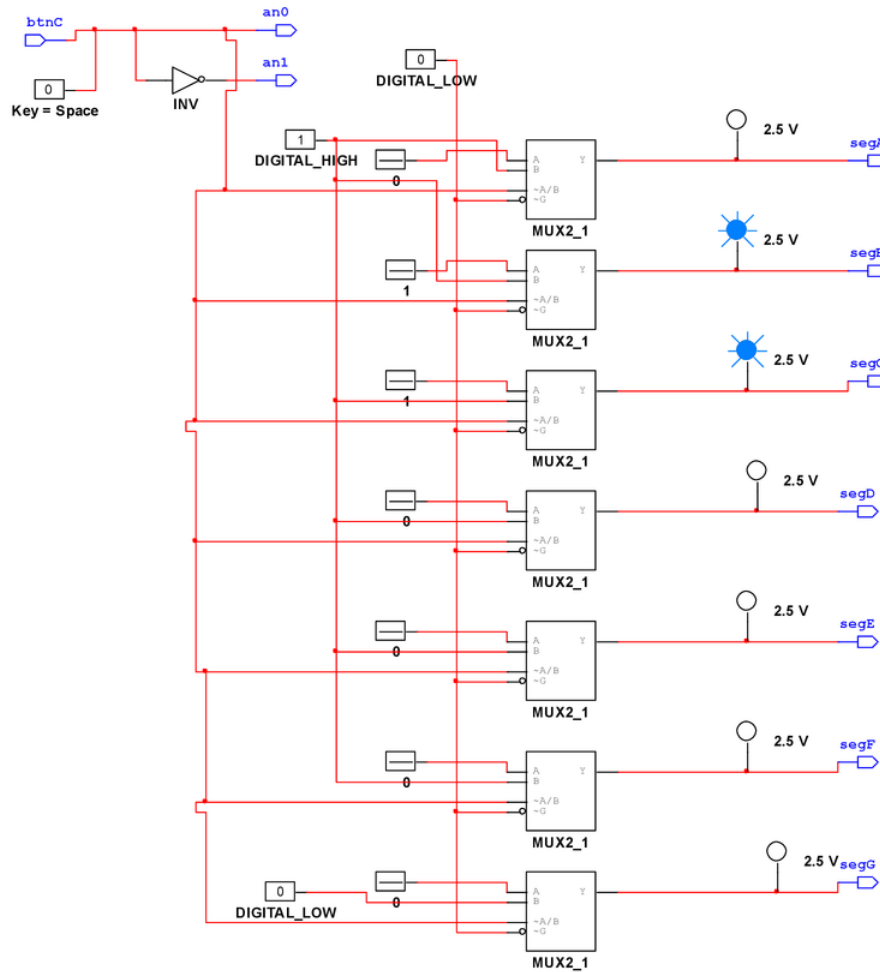


Figure 19 - MUX TENS

Before downloading to the board use of the interactive constant set to space bar was made to switch between TENS (as above) which will display the number 1 and the UNITS (as below) which will always display a 0.

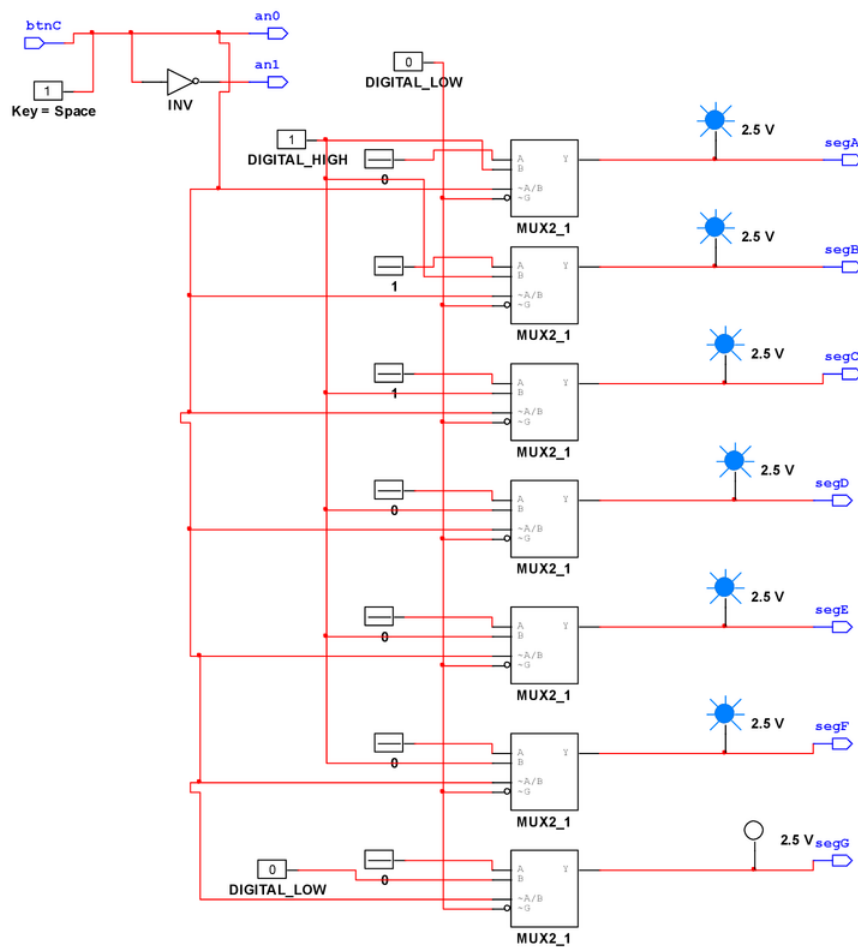


Figure 20 - MUX UNITS

The next step is to use the digital clock as shown below and set to 1Hz to witness the change happening automatically.

The next step is to use the centre button on the board to alternate between displays.

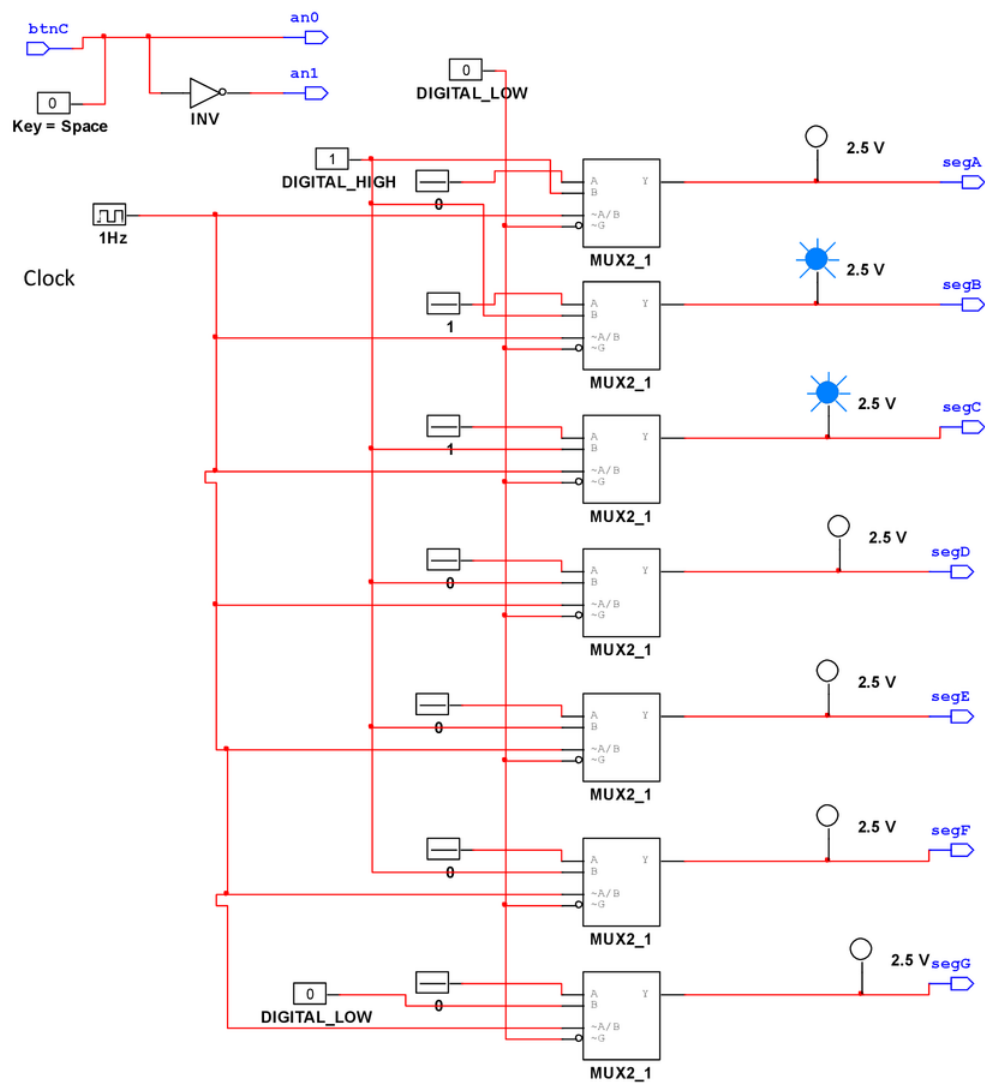


Figure 21 - Clocked MUX TENS

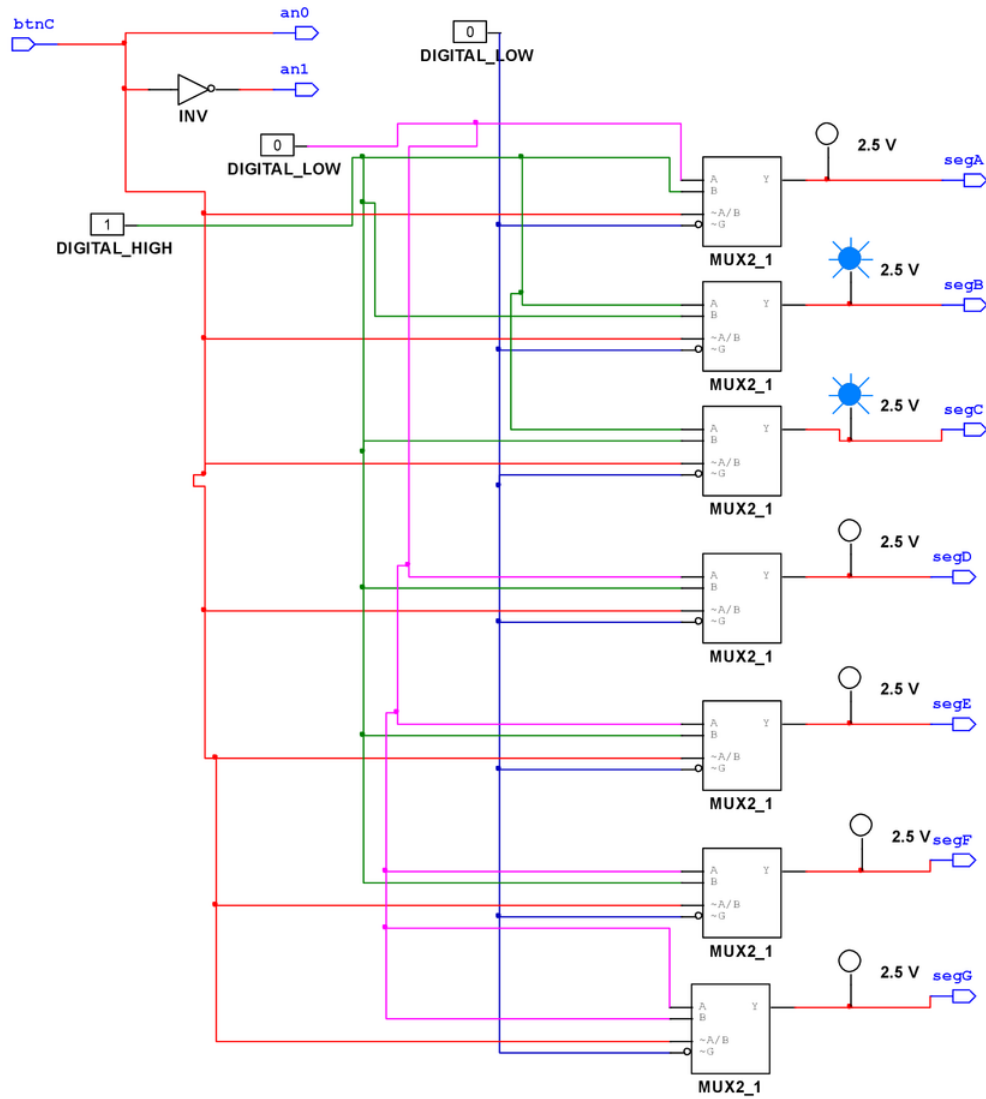


Figure 22 - MUX logic levels

When downloading to board all interactive digital constants were removed and replaced with digital Low or High as appropriate.

The testing of the MUX circuit was successful on multisim.

In order to complete the design the multiplexer was added as a sub circuit.

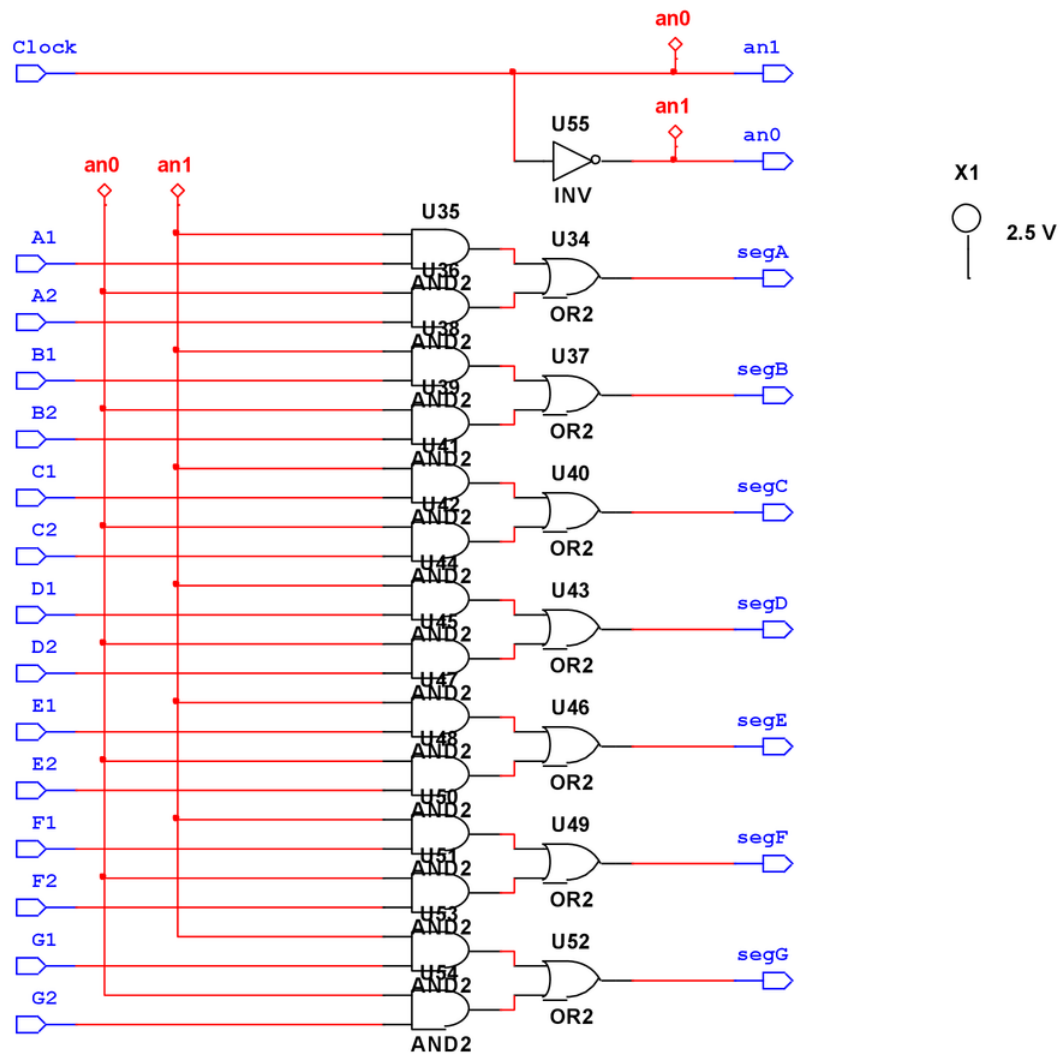


Figure 23 - MUX subcircuit logic

For a third segment to display the letter P then the following gates are needed on each segment, Fig 21 shows this just for segment G.

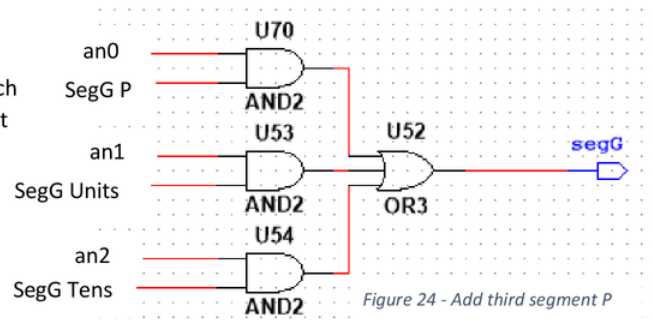


Figure 24 - Add third segment P

As shown below the 0 for the UNITS seven segment display has been added with 000 on pins ABC as constants to set the display to 0.

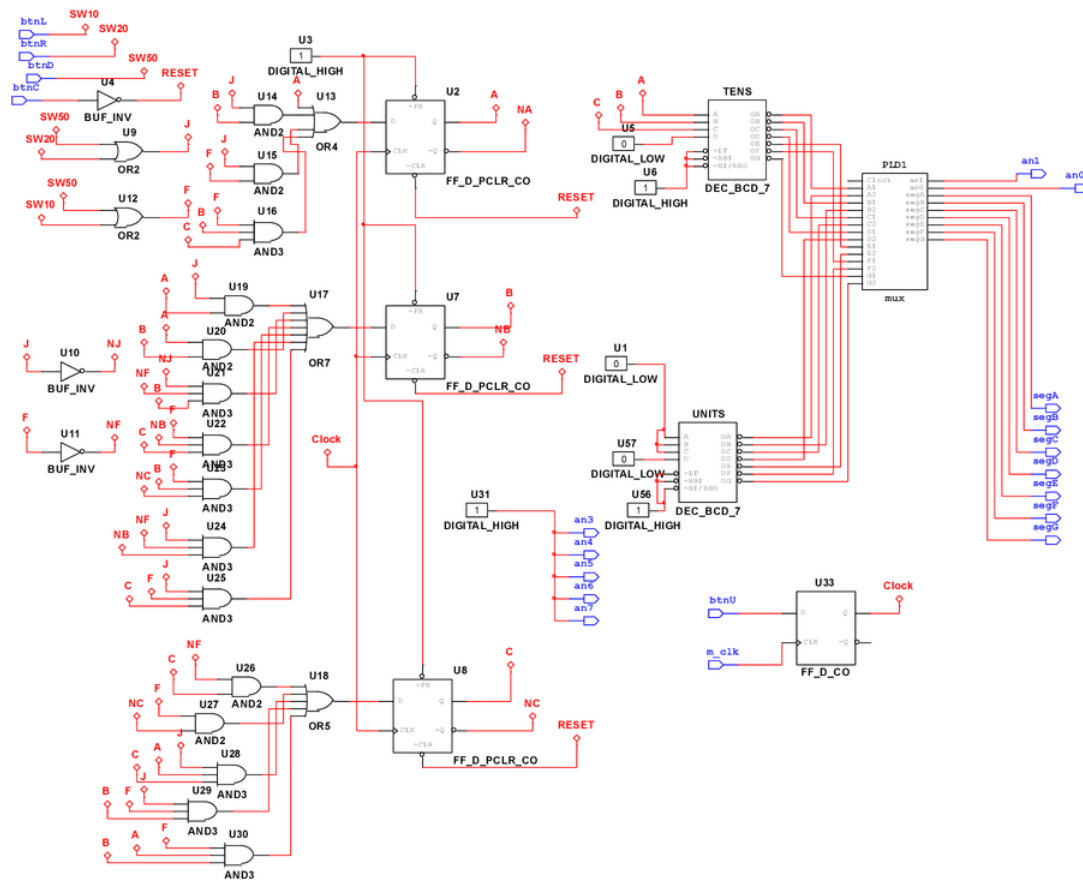


Figure 25 - PLD test 2

The next step is to figure out a process to slow down the master clock to a 64Hz clock.

This is achieved with *m_clk*, which must be slowed down by using an external signal generator connected to JA1 set to 64Hz.

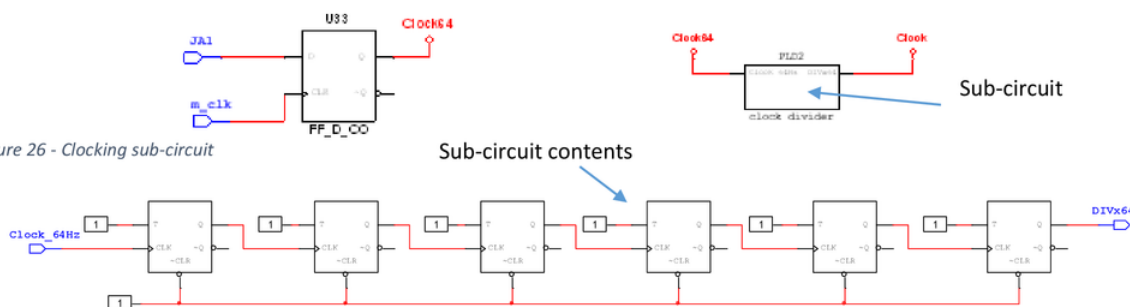
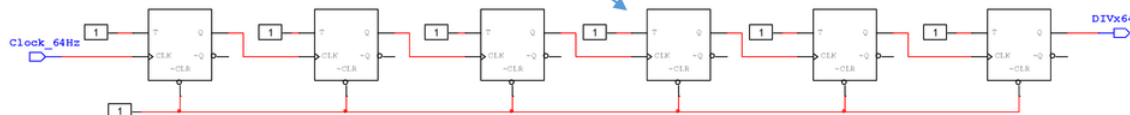
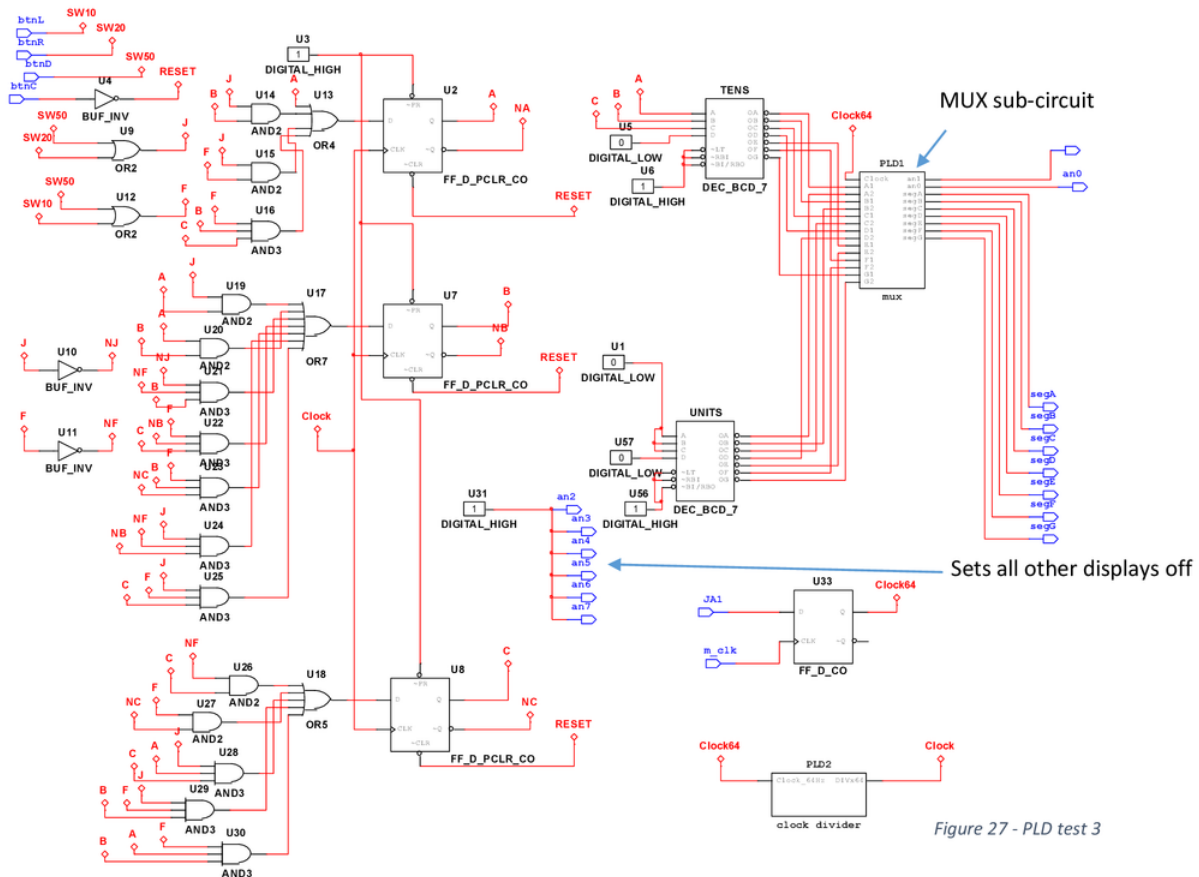


Figure 26 - Clocking sub-circuit

Sub-circuit contents



As shown above a divider circuit which takes 64 Hz and splits it to 32, 16, 8, 4, 2 and finally 1 Hz is utilised.



Initially when tested the displays were jumping to the wrong numbers, so a 1 was a 4 and a 3 was a 5 etc.

By swapping the ABC round the numbers were correct.

A	B	C	Decimal
0	0	1	1
1	0	0	4
0	1	1	3
1	1	0	5



These shots show the final downloaded version working correctly.

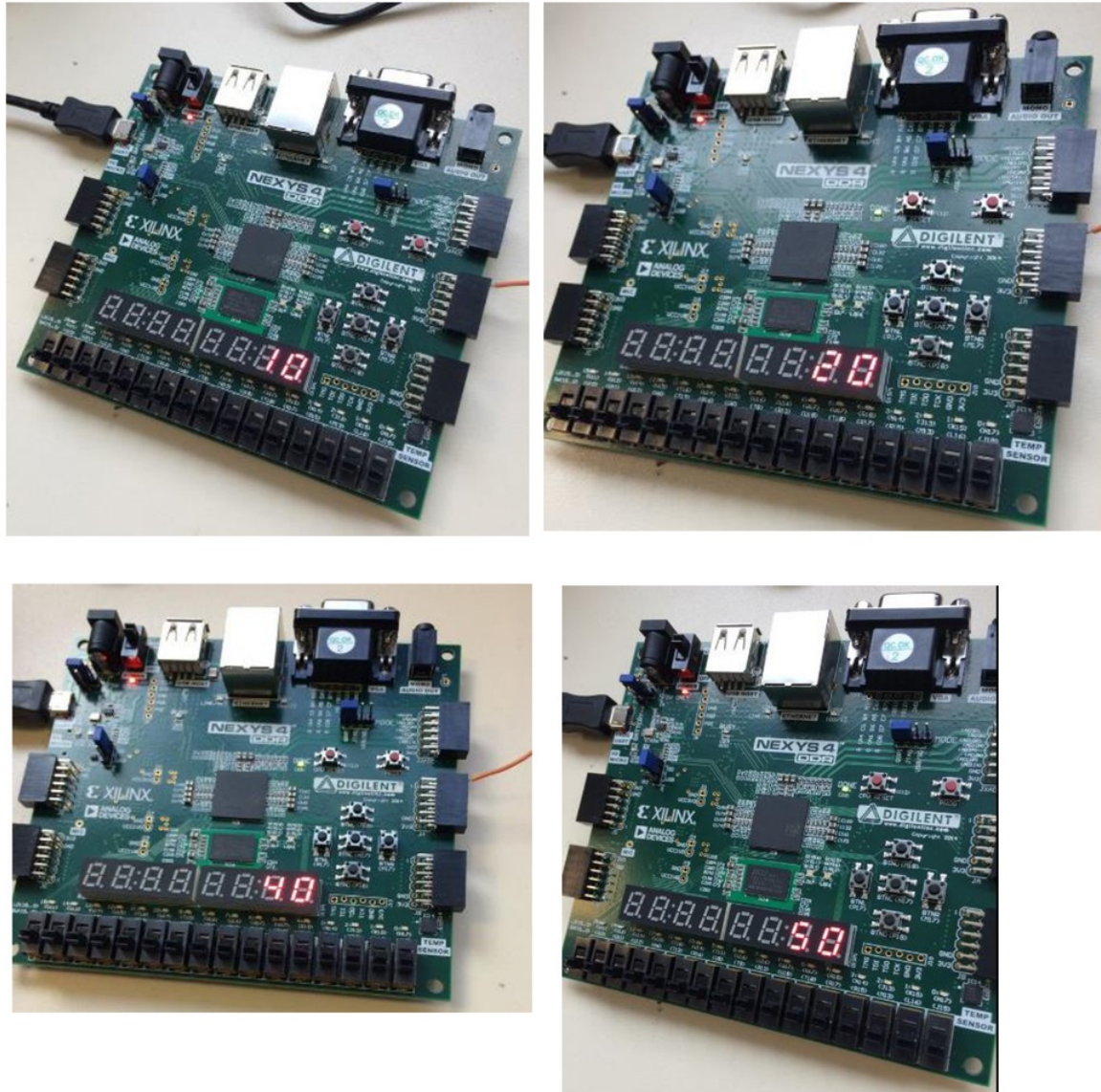


Figure 29 - Board on test

4.3 VHDL Design in Vivadoo

VHDL code was obtained to achieve the vending machine design, (full code given in appendix A) some of the code was missing and had to be written. A portion is detailed here.

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

entity VM1_2017 is
port (
    Seventy          : out std_logic;
    Block_10, Block_20, Block_50 : out std_logic;
    ten, twenty, fifty : in std_logic;
    reset             : in std_logic;
    Y                 : out STD_LOGIC_VECTOR (13 downto 0)
);
end VM1_2017;
```

Figure 31 - VM1 Entity

Figure 30 - Entity for 70

Following the necessary headers for file location the entity is defined with appropriate inputs and outputs.

Then the architecture is defined and the FSM can be set up to respond to the clock pulses.

Each state can be written to reflect the state diagram design

```
when S0 =>
    Seventy      <= '0';
    Block_10     <= '0';
    Block_20     <= '0';
    Block_50     <= '0';

    if (ten='0' and twenty='0' and fifty='0') then
        n_s <= S0;
    elsif (ten='1') then
        n_s <= S1;
    elsif (twenty='1') then
        n_s <= S2;
    elsif (fifty='1') then
        n_s <= S5;
    end if;
```

Figure 33 - Example code for 70

and provide the correct next state

sequence. As an example the 'if else' statements here show the progression from state zero.

This code needs to be correct for each state in the FSM.

Refer to full code in appendix A.

```
architecture Behavioral of VM1_2017 is
    -- providing FSM clock input
    signal Clk : std_logic;
    --type of state machine and signal declaration.
    type state_type is (S0,S1,S2,S3,S4,S5,S6,S7);
    signal p_s,n_s : state_type;
```

Figure 32 - VM1 Architecture

VHDL testing

The testing of the design was achieved via a test bench file (contained in appendix A) then by running a simulation. As can be seen below the results reflected the correct operation sequence.

Simulation result (10,10,10,10,10,10,10)

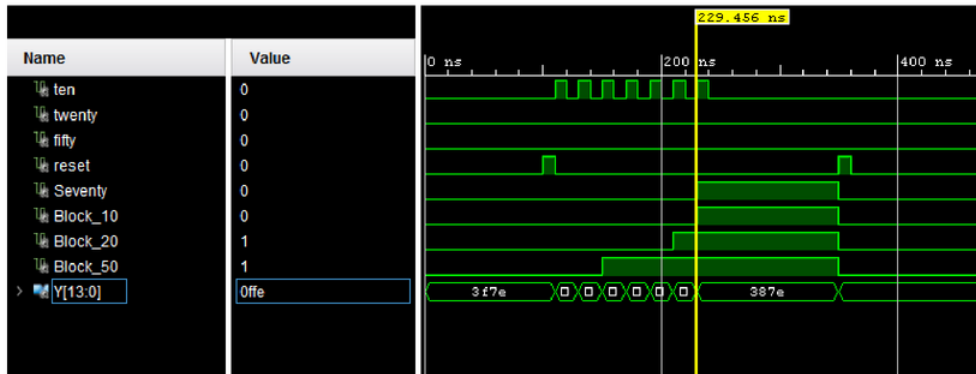


Figure 34 - Simulation test 1

The result of seventy can be seen to go high at the correct input value, also the block facility is seen to go high when the coin entered would exceed the maximum value of 70.

Simulation result (10,20,20,10,10)

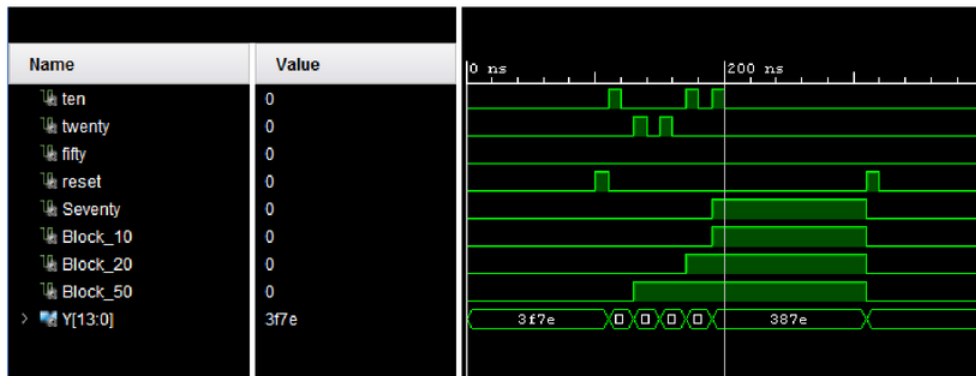


Figure 35 - Simulation test 2

Again, the seventy switches high when the correct money has been entered and the blocking facility is working correctly. If the user enters more than the required amount a mechanical lever will operate and divert the coin to eject it.

Simulation results (10, 50, 10)

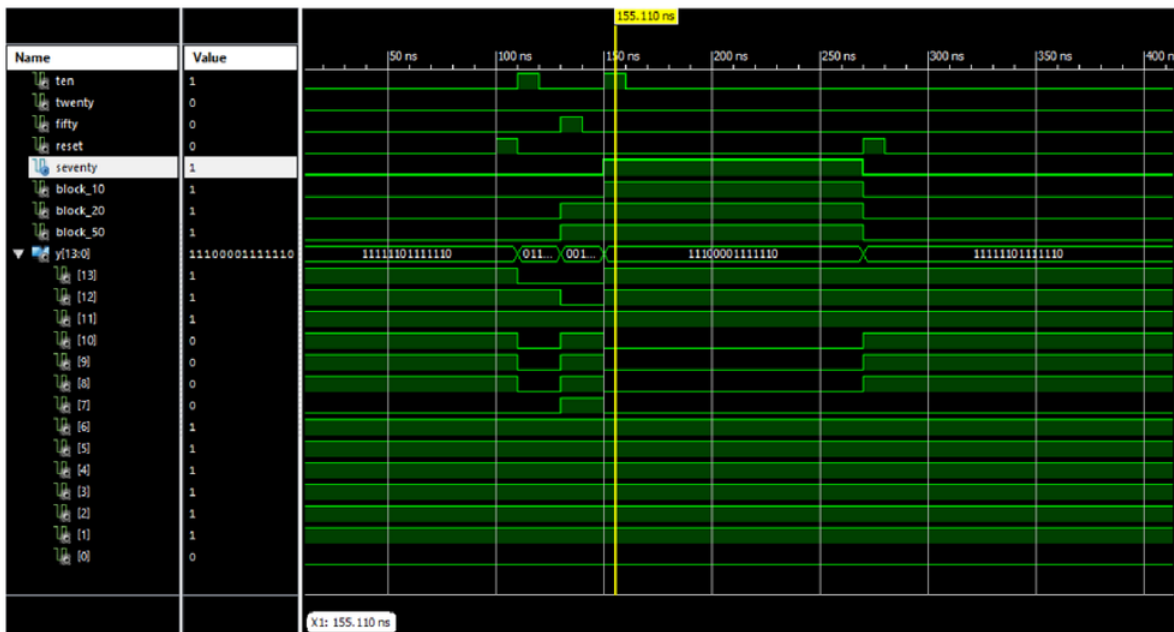


Figure 36 - Simulation test 3

This time the sequence operates and blocks both 20 and 50 upon reaching 60p, then when 70p is reached the output goes high and would release the product via a mechanical relay. By expanding the simulation each line can be seen with the BCD above which corresponds to the displayed value.

4.4 Adaptation in VHDL

Adaptation for 90p product is achieved easily by modifying the code in VHDL, starting with the entity definition, which is modified, for a ninety output.

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

entity VM2_2017 is
port(
    Seventy, Ninety          :    inout std_logic;
    Release                  :    inout std_logic;
    Block_10,Block_20,Block_50 :    out std_logic;
    ten,twenty,fifty        :    in std_logic;
    reset                   :    in std_logic;
    A,B,C,D                :    in std_logic;
    E,F,G,H                :    in std_logic;
    Y                      :    out STD_LOGIC_VECTOR (13 downto 0)
);
end VM2_2017;
```

Figure 37 – VM2 Entity

The architecture now reflects the extra states needed for this modification.

```
when S9 =>
    Seventy<='0';
    Ninety <='1';
    Block_10 <='1';
    Block_20 <='1';
    Block_50 <='1';
    if (ten='0' and twenty='0' and fifty='0')
        then
            n_s <= S9;
        else
            if (ten='1') then
                n_s <= S9;
            elsif (twenty='1') then
                n_s <= S9;
            elsif (fifty='1') then
                n_s <= S9;
            end if;
        end case;
    end process;

-- Reset conditions
Sig1 <= A or B or C or D;
Sig2 <= E or F or G or H;
Release <= (Sig1 and Seventy) or (Sig2 and Ninety);

--Outputing to the 2*7 segment displays
with p_s select
    Y <= "11111101111110" when S0,
        "01100001111110" when S1,
        "11011011111110" when S2,
        "11110011111110" when S3,
        "01100111111110" when S4,
        "10110111111110" when S5,
        "00111111111110" when S6,
        "11100001111110" when S7,
        "11111111111110" when S8,
        "11100111111110" when S9;

end Behavioral;
```

architecture Behavioral of VM2_2017 is

```
signal Sig1      :    std_logic;
signal Sig2      :    std_logic;
signal Clk       :    std_logic;
```

--type of state machine and signal declaration.

```
type state_type is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9);
signal p_s,n_s : state_type;
```

Figure 38 – VM2 Architecture

Example of the code shows operation for state 9 and the corresponding output actions including Sig1 and Sig2 for the release of the corresponding value product.

Figure 39 - Example code for 90

The full VHDL code for the adaptation and the test bench file is contained in appendix B.

Simulated results were used to thoroughly test the design.

Simulation Result (50, 10, 10, 10, 10)

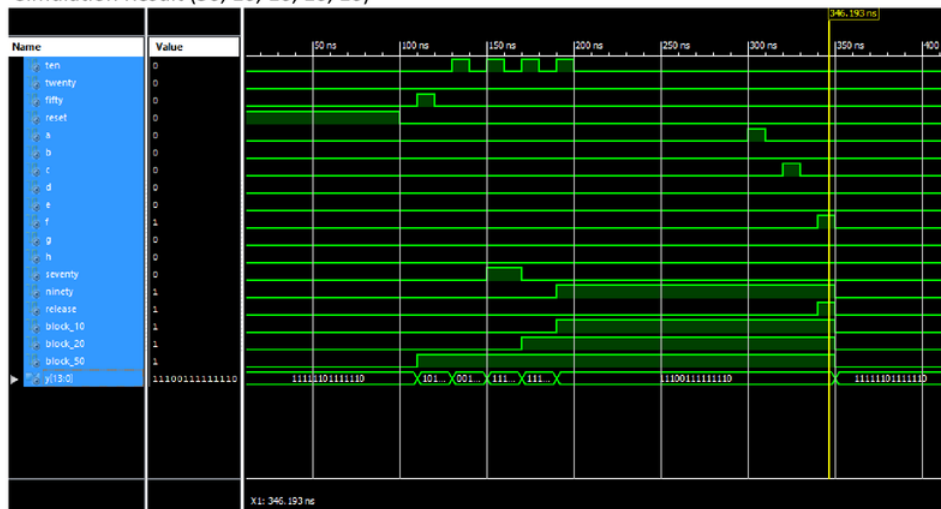


Figure 40 - Simulation test 4

Both seventy and then ninety outputs are seen to go high upon reaching the corresponding value and the operation of the block facility is also correct.

Simulation Result (20, 20, 20, 10)

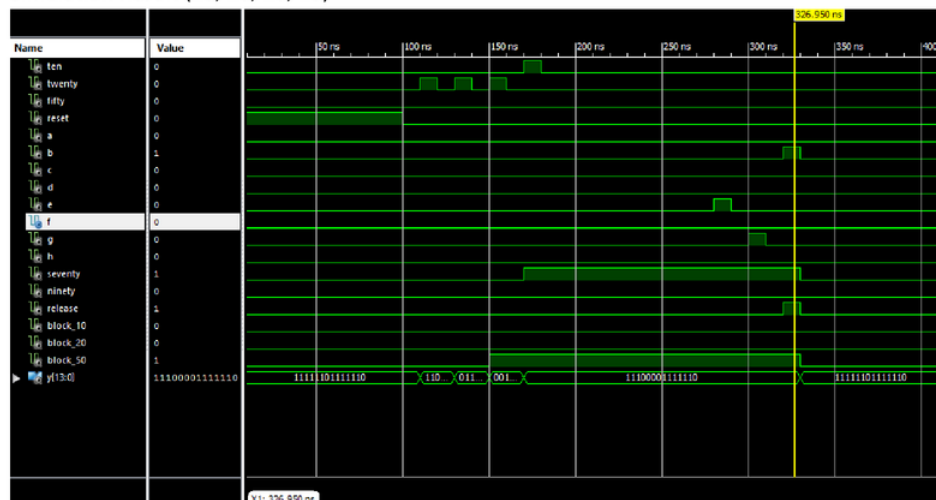


Figure 41 - Simulation test 5

This test shows the seventy output operating but the ninety output stays low as insufficient funds inserted.

Expanding the above test to show the seven segment displays for 0p, 20p, 40p, 60p, 70p, E then G operated with no release then B operated which outputs and resets back to state 0.

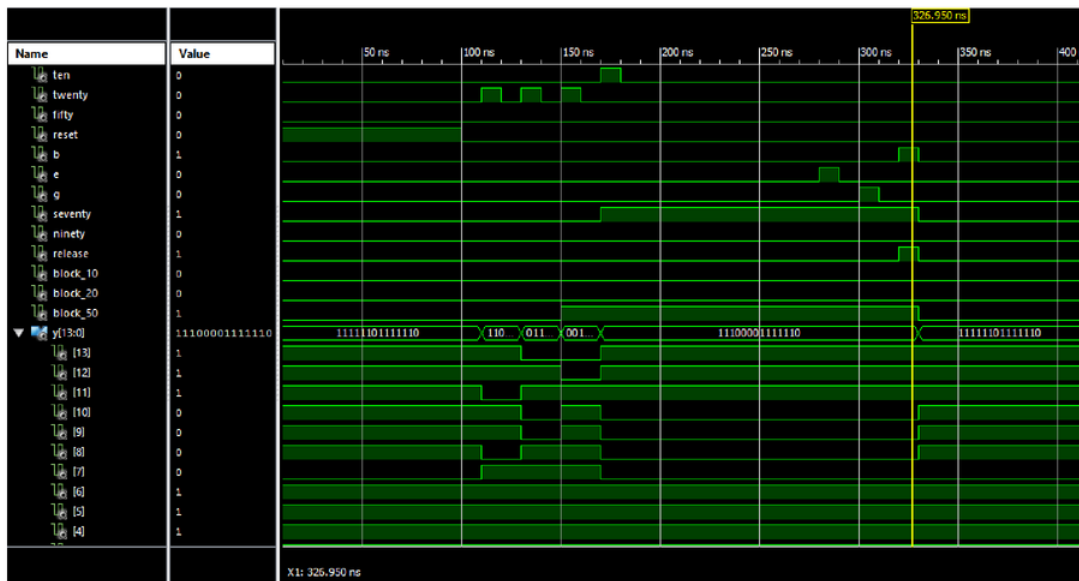


Figure 42 - Simulation test 6

It can be seen that adapting the code is a quick and easy thing to do, whereas adaptation of the multisim would have taken much more time and effort, due to the complexity of the logic involved a start over approach would have been needed.

5 Conclusion

In conclusion, comparison of the implementation methods and design considerations featured heavily in the choices made throughout the development process.

Demonstrating an understanding of advanced design methods and current implementation of technical issues within digital electronics has proved successful and this is verified with the testing procedures and results.

A vending machine control system from customer specification and FSM design was developed using two contrasting methods.

A detailed design specification allowed the use of a 'Moore' model FSM and subsequent construction and test. The design consisted of the control aspects and display requirements, which were again broken down into clocking through a FSM then decoding the outputs and multiplexing for display purposes.

Points 1 and 2 for VHDL are considerable factors, this is relatively new technology and so initial research had to be carried out and the Vivadoo software learned via books and internet. There are several tutorials available on-line which made this process a lot simpler. In contrast multisim is known and simple to use. On point 3 VHDL wins as once key code blocks were formed they were used repeatedly. Point 4 is again a strong advantage for VHDL as discussed in section 4.4 Point 5 sees advantages for both methods, the key advantage of VHDL in system design is allowing system behaviour to be modelled and verified via simulation before translating the design onto hardware.

Design consideration	Multisim		VHDL	
	Advantage	Disadvantage	Advantage	Disadvantage
1. The level of knowledge required	This is a familiar package and is simple to use	PLD version is not so familiar	Lots of tutorials available on-line	A relatively new platform significant knowledge required
2. The learning curve inclination	Not a steep learning curve	None	Once learned potential is large	Quite a steep learning curve
3. Speed of implementation	Sometimes copy and paste facility is useful	Lengthy construction times for complex circuits	Library of code blocks available	Initial coding of blocks takes time
4. Ability to modify and change the specification	Changes in component values are quick to do	Long lead time when making circuit changes	Simple and quick to modify	None
5. Testing robustness	Logic analyser available	Downloading to FPGA takes time	Simulation function tests logic	Formation of test bench needed

The preferred method of design in terms of ease of adaptation is the VHDL design, as can be seen in section 4.4 the code is very quick and easy to adapt.

In conclusion, results obtained matched the customer specification and a low cost solution reached.

6 References

Readler, B. (2014). *VHDL by example*. [SI]: Full Arc Press.

Srmuniv.ac.in. (2018). [online] Available at:
<http://www.srmuniv.ac.in/ramapuram/sites/ramapuram/files/EC308.pdf> [Accessed 23 Mar. 2018].

Reference.digilentinc.com. (2018). *Nexys 4 DDR Reference Manual [Reference.Digilentinc]*. [online]
Available at: <https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual> [Accessed 26 Mar. 2018].

Xilinx.com. (2018). *Digilent Nexys 4 Artix-7 FPGA Trainer Board*. [online] Available at:
<https://www.xilinx.com/products/boards-and-kits/1-3yznp5.html> [Accessed 26 Mar. 2018].

cards, I., Card, U. and Card, U. (2018). *User Manual for Seven Segment Display Card*. [online]
Pantechsolutions.net. Available at: <https://www.pantechsolutions.net/interface-cards-tutorials/user-manual-for-seven-segment-display-card> [Accessed 26 Mar. 2018].

Appendix A

VHDL code for 70p input vending machine

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

entity VM1_2017 is
port (
    Seventy          : out std_logic;
    Block_10, Block_20, Block_50 : out std_logic;
    ten, twenty, fifty : in std_logic;
    reset            : in std_logic;
    Y                : out STD_LOGIC_VECTOR (13 downto 0)
);

end VM1_2017;

architecture Behavioral of VM1_2017 is
-- providing FSM clock input
signal Clk      : std_logic;
--type of state machine and signal declaration.
type state_type is (S0,S1,S2,S3,S4,S5,S6,S7);
signal p_s,n_s : state_type;

begin
-- FSM clock is set if a 10 or 20 or 50p is inserted

Clk <= ten or twenty or fifty;
```

--Finite state machine set-up

```
process(Clk, reset)
```

```
begin
```

```
    if (reset = '1') then
```

```
        p_s <= S0; --default state on reset.
```

```
    elsif(rising_edge(Clk)) then
```

```
        p_s <= n_s;
```

```
    end if;
```

```
end process;
```

--8 stage state machine

```
process (p_s, n_s, ten, twenty, fifty)
```

```
begin
```

```
case p_s is
```

```
    when S0 =>
```

```
        Seventy          <= '0';
```

```
        Block_10         <= '0';
```

```
        Block_20         <= '0';
```

```
        Block_50         <= '0';
```

```
    if      (ten='0' and twenty='0' and fifty='0')  then
```

```
        n_s <= S0;
```

```
    elsif (ten='1')  then
```

```
        n_s <= S1;
```

```
    elsif (twenty='1')  then
```

```

                                n_s <= S2;
                                elsif (fifty='1')      then
                                n_s <= S5;

                                end if;

when S1 =>

    Seventy    <= '0';
    Block_10   <= '0';
    Block_20   <= '0';
    Block_50   <= '0';

    if (ten='0' and twenty='0' and fifty='0') then
        n_s <= S1;
        elsif (ten='1') then
            n_s <= S2;
        elsif (twenty='1') then
            n_s <= S3;
        elsif (fifty='1') then
            n_s <= S5;
        end if;

when S2 =>

    Seventy    <= '0';
    Block_10   <= '0';
    Block_20   <= '0';
    Block_50   <= '0';

    if (ten='0' and twenty='0' and fifty='0') then
        n_s <= S2;

```

```

elseif (ten='1')    then
n_s <= S3;
elseif (twenty='1') then
n_s <= S4;
elseif (fifty='1')  then
n_s <= S7;
end if;

```

when S3 =>

```

Seventy    <= '0';
Block_10   <= '0';
Block_20   <= '0';
Block_50   <= '1';

```

```

if (ten='0' and twenty='0' and fifty='0') then
n_s <= S3;
elseif (ten='1')    then
n_s <= S4;
elseif (twenty='1') then
n_s <= S5;
elseif (fifty='1')  then
n_s <= S3;
end if;

```

when S4 =>

```

Seventy    <= '0';
Block_10   <= '0';
Block_20   <= '0';
Block_50   <= '1';

```

```

if (ten='0' and twenty='0' and fifty='0') then

```

```

n_s <= S4;
elsif (ten='1') then
n_s <= S5;
elsif (twenty='1') then
n_s <= S6;
elsif (fifty='1') then
n_s <= S4;
end if;

when S5 =>

```

```

Seventy    <= '0';
Block_10   <= '0';
Block_20   <= '0';
Block_50   <= '1';

```

```

if (ten='0' and twenty='0' and fifty='0') then
n_s <= S5;
elsif (ten='1') then
n_s <= S6;
elsif (twenty='1') then
n_s <= S7;
elsif (fifty='1') then
n_s <= S5;
end if;

when S6 =>

```

```

Seventy    <= '0';
Block_10   <= '0';
Block_20   <= '1';
Block_50   <= '1';

```

```

        if (ten='0' and twenty='0' and fifty='0') then
            n_s <= S6;
        elsif (ten='1') then
            n_s <= S7;
        elsif (twenty='1') then
            n_s <= S6;
        elsif (fifty='1') then
            n_s <= S6;
        end if;

when S7 =>

        Seventy <= '1';
        Block_10 <= '1';
        Block_20 <= '1';
        Block_50 <= '1';

then

        if (ten='0' and twenty='0' and fifty='0')

            n_s <= S7;
        elsif (ten='1') then
            n_s <= S7;
        elsif (twenty='1') then
            n_s <= S7;
        elsif (fifty='1') then
            n_s <= S7;
        end if;

        end case;
    end process;

```

-- Outputting to both seven segment displays

with p_s select

```
Y      <=  "11111101111110" when S0,  
           "01100001111110" when S1,  
           "11011011111110" when S2,  
           "11110011111110" when S3,  
           "01100111111110" when S4,  
           "10110111111110" when S5,  
           "00111111111110" when S6,  
           "11100001111110" when S7;
```

end Behavioral;

Test Bench stimulus

```
-- Stimulus process
stim_proc: process
    -- insert stimulus here
begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

    reset <= '1';
    wait for 10ns;
    reset <= '0';

    ten <= '1';
    wait for 10ns;
    ten <= '0';

    wait for 10ns;
    ten <= '1';
    wait for 10ns;
    ten <= '0';

    wait for 10ns;
    ten <= '1';
    wait for 10ns;
    ten <= '0';
```

```
wait for 10ns;  
ten <= '1';  
wait for 10ns;  
ten <= '0';
```

```
wait for 10ns;  
ten <= '1';  
wait for 10ns;  
ten <= '0';
```

```
wait for 10ns;  
ten <= '1';  
wait for 10ns;  
ten <= '0';
```

```
wait for 10ns;  
ten <= '1';  
wait for 10ns;  
ten <= '0';
```

```
wait for 110ns;  
reset <= '1';  
wait for 10ns;  
reset <= '0';
```

```
wait;  
end process;
```

```
END;
```

Appendix B

VHDL code for 90p input vending machine

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

entity VM2_2017 is
port(
    Seventy, Ninety          : inout std_logic;
    Release                  : inout std_logic;
    Block_10,Block_20,Block_50 : out std_logic;
    ten,twenty,fifty         : in std_logic;
    reset                    : in std_logic;
    A,B,C,D                  : in std_logic;
    E,F,G,H                  : in std_logic;
    Y                        : out STD_LOGIC_VECTOR (13 downto 0)
);
end VM2_2017;

architecture Behavioral of VM2_2017 is

    signal Sig1      : std_logic;
    signal Sig2      : std_logic;
    signal Clk        : std_logic;

    --type of state machine and signal declaration.

    type state_type is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9);
    signal p_s,n_s : state_type;

begin

    -- Setting up FSM clock

    Clk    <= ten or twenty or fifty;

    --Finite state machine set-up

    process(Clk, reset, release)
    begin
        if (reset = '1' or falling_edge (release)) then
            p_s <= S0; --default state on reset.
        elsif(rising_edge(Clk)) then
            p_s <= n_s;
        end if;
    end process;
```

--Ten stage state machine

```
process (p_s, n_s, ten, twenty, fifty)
begin

case p_s is

when S0 =>

                                Seventy<= '0';
                                Ninety  <= '0';
                                Block_10    <= '0';
                                Block_20    <= '0';
                                Block_50 <= '0';
if    (ten='0' and twenty='0' and fifty='0')    then
    n_s <= S0;
                                elsif (ten='1')    then
    n_s <= S1;
                                elsif (twenty='1')    then
    n_s <= S2;
                                elsif (fifty='1')    then
    n_s <= S5;
end if;

                                when S1 =>

                                Seventy<= '0';
                                Ninety  <= '0';
                                Block_10    <= '0';
                                Block_20    <= '0';
                                Block_50 <= '0';
if    (ten='0' and twenty='0' and fifty='0')    then
    n_s <= S1;
                                elsif (ten='1')    then
    n_s <= S2;
                                elsif (twenty='1')    then
    n_s <= S3;
                                elsif (fifty='1')    then
    n_s <= S6;
end if;

                                when S2 =>

                                Seventy<= '0';
                                Ninety  <= '0';
                                Block_10    <= '0';
                                Block_20    <= '0';
                                Block_50 <= '0';
if    (ten='0' and twenty='0' and fifty='0')    then
    n_s <= S2;
                                elsif (ten='1')    then
```

```

n_s <= S3;
elseif (twenty='1') then
n_s <= S4;
elseif (fifty='1') then
n_s <= S7;
end if;

when S3 =>
    Seventy<= '0';
    Ninety <= '0';
    Block_10 <= '0';
    Block_20 <= '0';
    Block_50 <= '0';
if (ten='0' and twenty='0' and fifty='0') then
n_s <= S3;
elseif (ten='1') then
n_s <= S4;
elseif (twenty='1') then
n_s <= S5;
elseif (fifty='1') then
n_s <= S8;
end if;

when S4 =>
    Seventy<= '0';
    Ninety <= '0';
    Block_10 <= '0';
    Block_20 <= '0';
    Block_50 <= '0';
if (ten='0' and twenty='0' and fifty='0') then
n_s <= S4;
elseif (ten='1') then
n_s <= S5;
elseif (twenty='1') then
n_s <= S6;
elseif (fifty='1') then
n_s <= S9;
end if;

when S5 =>
    Seventy<= '0';
    Ninety <= '0';
    Block_10 <= '0';
    Block_20 <= '0';
    Block_50 <= '1';
if (ten='0' and twenty='0' and fifty='0') then
n_s <= S5;
elseif (ten='1') then
n_s <= S6;
elseif (twenty='1') then
n_s <= S7;

```

```

        elsif (fifty='1') then
            n_s <= S9;

end if;

        when S6 =>
            Seventy<= '0';
            Ninety <= '0';
            Block_10 <= '0';
            Block_20 <= '0';
            Block_50 <= '1';
if      (ten='0' and twenty='0' and fifty='0') then
    n_s <= S6;
        elsif (ten='1') then
            n_s <= S7;
        elsif (twenty='1') then
            n_s <= S8;
        elsif (fifty='1') then
            n_s <= S9;

end if;

        when S7 =>
            Seventy<= '1';
            Ninety <= '0';
            Block_10 <= '0';
            Block_20 <= '0';
            Block_50 <= '1';
if      (ten='0' and twenty='0' and fifty='0') then
    n_s <= S7;
        elsif (ten='1') then
            n_s <= S8;
        elsif (twenty='1') then
            n_s <= S9;
        elsif (fifty='1') then
            n_s <= S9;

end if;

        when S8 =>
            Seventy<= '0';
            Ninety <= '0';
            Block_10 <= '0';
            Block_20 <= '1';
            Block_50 <= '1';
if      (ten='0' and twenty='0' and fifty='0') then
    n_s <= S8;
        elsif (ten='1') then
            n_s <= S9;
        elsif (twenty='1') then
            n_s <= S9;

```

```

                                elsif (fifty='1') then
                                    n_s <= S9;
                                end if;

                                when S9 =>
                                    Seventy<= '0';
                                    Ninety <= '1';
                                    Block_10    <= '1';
                                    Block_20    <= '1';
                                    Block_50 <= '1';
                                    if          (ten='0' and twenty='0' and fifty='0')

                                        elsif (ten='1') then
                                            n_s <= S9;
                                        elsif (twenty='1') then
                                            n_s <= S9;
                                        elsif (fifty='1') then
                                            n_s <= S9;
                                        end if;
                                    end case;
                                end process;

                                -- Reset conditions

                                Sig1      <=      A or B or C or D;
                                Sig2      <=      E or F or G or H;
                                Release   <=      (Sig1 and Seventy) or (Sig2 and Ninety);

                                --Outputing to the 2*7 segment displays

                                with p_s select

                                    Y      <=      "11111101111110" when S0,
                                                    "01100001111110" when S1,
                                                    "11011011111110" when S2,
                                                    "11110011111110" when S3,
                                                    "01100111111110" when S4,
                                                    "10110111111110" when S5,
                                                    "00111111111110" when S6,
                                                    "11100001111110" when S7,
                                                    "11111111111110" when S8,
                                                    "11100111111110" when S9;

                                end Behavioral;

```

Test Bench stimulus

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY VM2_2017_tb IS
END VM2_2017_tb;

ARCHITECTURE behavior OF VM2_2017_tb IS

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

    COMPONENT VM2_2017
    PORT(

        Seventy : INOUT std_logic;
        Ninety : INOUT std_logic;
                Release : INOUT std_logic;
                Block_10 : OUT std_logic;
                Block_20 : OUT std_logic;
                Block_50 : OUT std_logic;
        ten : IN std_logic;
        twenty : IN std_logic;
                fifty : IN std_logic;
                reset : IN std_logic;
                A,B,C,D : IN std_logic;
                E,F,G,H : IN std_logic;
        Y : OUT std_logic_vector(13 downto 0)
    );
    END COMPONENT;

    --Inputs

    signal ten : std_logic := '0';
    signal twenty : std_logic := '0';
        signal fifty : std_logic := '0';
        signal reset : std_logic := '1';
        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';
    signal F : std_logic := '0';
    signal G : std_logic := '0';
    signal H : std_logic := '0';

    --Outputs
    signal Seventy : std_logic;
    signal Ninety : std_logic;
    signal Release : std_logic;
    signal Block_10 : std_logic;
    signal Block_20 : std_logic;
    signal Block_50 : std_logic;
    signal Y : std_logic_vector(13 downto 0);

    -- Clock period definitions

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: VM2_2017 PORT MAP (

        Seventy => Seventy,
        Ninety => Ninety,
        Release => Release,
        Block_10 => Block_10,
        Block_20 => Block_20,
        Block_50 => Block_50,

        ten => ten,
        twenty => twenty,
        fifty => fifty,
        reset => reset,
        A => A,
        B => B,
        C => C,
        D => D,
        E => E,
        F => F,
        G => G,
        H => H,

        Y => Y
    );

    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        wait for 100 ns;

        reset <= '0';

```

```
wait for 10ns;
fifty <='1';
wait for 10ns;
fifty <='0';
wait for 10ns;
ten <= '1';
wait for 10ns;
ten <= '0';
wait for 10ns;
ten <= '1';
wait for 10ns;
ten <= '0';
wait for 10ns;
ten <= '1';
wait for 10ns;
ten <= '0';
wait for 100ns;
A <= '1';
wait for 10ns;
A <= '0';
wait for 10ns;
C <= '1';
wait for 10ns;
C <= '0';
wait for 10ns;
F <= '1';
wait for 10ns;
F <= '0';

wait;
end process;

END;
```

assignment 2

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

87 /100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

LIESEL WILLIAMS**87****RESEARCH-INFORMED LITERATURE**

There was evidence of extensive further reading and research from credible and reliable sources. Your interpretation and understanding of the additional research was very good and demonstrated a deep underlying appreciation of the subject. Your references were appropriate to the task and your reference list was of an appropriate length for this assignment.

8**KNOWLEDGE AND UNDERSTANDING**

The knowledge and understanding of the subject demonstrated by this assignment was generally excellent. There were very few gaps in your knowledge of the basic principles and the higher level concepts were very well explained. Your explanation of the design process and calculations was also excellent and it did extend your knowledge well beyond that contained in the course notes.

27**ANALYSIS**

Your analysis of the task was excellent, both in class and as shown in this assignment. You have demonstrated very good analytical skills and your approach to the task was very logical. You carried

18

out a detailed evaluation of the results and included an excellent conclusion relating your practical results to accepted theory.

PRACTICAL APPLICATION AND DEP.



Your application of practical skills for this task was generally excellent. You used the resources available in MultiSim appropriately when designing the circuit and you used appropriate options for testing and evaluating the circuit performance. You managed to design a very efficient circuit that functioned well within the specification and achieved with very little assistance. Your results were very detailed, well presented and allowed for an excellent conclusion to be attempted.

26

SKILLS FOR PROFESSIONAL PRAC.



In this assignment you have demonstrated skills in working individually and also collaboratively. Your written report was very well structured and written in a coherent and logical way. The report was presented in a professional manner using appropriate language and formatting, good diagrams in appropriate places, good sentence/paragraph structure and clear calculations

8