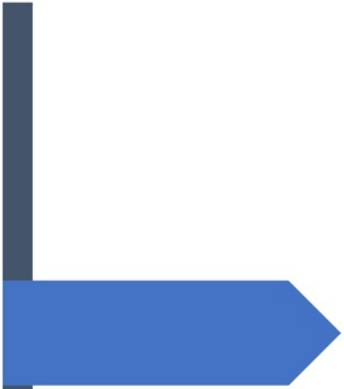


# Ass2 EDES final

*by* Michael Lewis

---

|                |                                                          |                 |        |
|----------------|----------------------------------------------------------|-----------------|--------|
| FILE           | 5293_MICHAEL_LEWIS_ASS2_EDES_FINAL_152853_536458812.DOCX |                 |        |
| TIME SUBMITTED | 09-APR-2018 03:27 PM (UTC+0100)                          | WORD COUNT      | 4590   |
| SUBMISSION ID  | 86235887                                                 | CHARACTER COUNT | 27 123 |



# PSDE 6015 – Electronic design and simulation

Assignment 2



Michael Lewis

## Contents

|                                                  |    |
|--------------------------------------------------|----|
| Table of figures .....                           | 2  |
| 1. Introduction, aims and objectives .....       | 3  |
| 2. Background research .....                     | 4  |
| Top down and bottom up design.....               | 4  |
| 3. Design method, choices and calculations ..... | 6  |
| State diagram.....                               | 6  |
| J and F signals.....                             | 7  |
| Truth table .....                                | 7  |
| Karnaugh map .....                               | 8  |
| Multisim circuit .....                           | 9  |
| PLD screenshot.....                              | 10 |
| 4. Testing and improving the design.....         | 12 |
| Multisim testing .....                           | 12 |
| Boolean expression testing evidence .....        | 13 |
| Multisim FPGA testing.....                       | 15 |
| VHDL testing.....                                | 16 |
| VHDL Nexys 4 testing .....                       | 19 |
| Further improving VHDL design.....               | 20 |
| 5. Conclusion.....                               | 23 |
| 6. Reference list and bibliography .....         | 25 |
| References .....                                 | 25 |
| 7. Appendix .....                                | 26 |
| VHDL code.....                                   | 26 |
| Testbench.....                                   | 31 |

## Table of figures

|                                                          |    |
|----------------------------------------------------------|----|
| Figure 1 - Simple vending machine .....                  | 3  |
| Figure 2 - Modified vending machine .....                | 3  |
| Figure 3 - top down and bottom up approach.....          | 4  |
| Figure 4 - Top down design flow chart .....              | 5  |
| Figure 5- Vending machine state diagram .....            | 6  |
| Figure 6 - J and F truth table .....                     | 7  |
| Figure 7 - Vending machine truth table .....             | 7  |
| Figure 8- Karnaugh Maps .....                            | 8  |
| Figure 9 - First multisim design .....                   | 10 |
| Figure 10 - Redesigned circuit in PLD mode .....         | 10 |
| Figure 11 - Clock divider.....                           | 11 |
| Figure 12 - Multiplexer.....                             | 11 |
| Figure 13 - Boolean expression test table.....           | 12 |
| Figure 14- Boolean testing 10p .....                     | 13 |
| Figure 15 - Boolean testing 20p .....                    | 13 |
| Figure 16 - Boolean testing 50p .....                    | 13 |
| Figure 17 - Boolean testing 70p and coin blocking .....  | 14 |
| Figure 18 - Boolean testing reset.....                   | 14 |
| Figure 19- Nexys 4 10, 20 and 50p testing.....           | 15 |
| Figure 20 - VHDL test 1 .....                            | 16 |
| Figure 21 - VHDL test 2 .....                            | 17 |
| Figure 22 - VHDL test 3 .....                            | 18 |
| Figure 23 - VHDL first design Nexys 4 board display..... | 19 |
| Figure 24 - Modified VHDL test 1.....                    | 21 |
| Figure 25- Modified VHDL test 2.....                     | 21 |
| Figure 26- Modified VHDL test 3.....                     | 22 |

## 1. Introduction, aims and objectives

The aims and objectives for this assignment are: To produce a Multisim PLD design of a simplified vending machine. Produce a VHDL design of the same vending machine system, both designs must be downloaded and tested on the Nexys 4 board.

Once both initial designs are complete, the VHDL design must be modified to suit a new specification. Both design methods must be critically evaluated and compared.

Figure 1 shows the simplified vending machine to be build using Vivadoo and Multisim, and eventually downloaded to the Nexys 4 board. Only 10, 20 and 50 Pence coins can be entered, with a maximum value of 70P entered into the machine. The FSM block either 50, 20, 10 coins if entering said coin would exceed 70P. It is assumed that 1 coin can be entered at once. The reset button will reset the machine to its starting state and set the money count to 0.

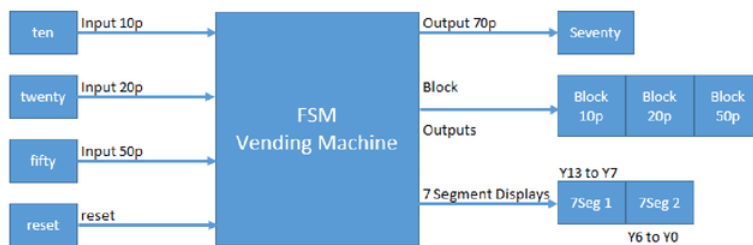


Figure 1 - Simple vending machine

The VHDL code for the simple vending machine must be adapted to suite a different purpose, see Figure 2. Effectively, the second system adds another 'item' to the pretend vending machine that costs 90P. As this has 2 items that can be purchased, the machine must activate the particular mech depending on the amount of money that is entered into the machine. Only correct sums of money are recognised and no change is given. The rest of the features from the first machine will remain for the modified FSM.

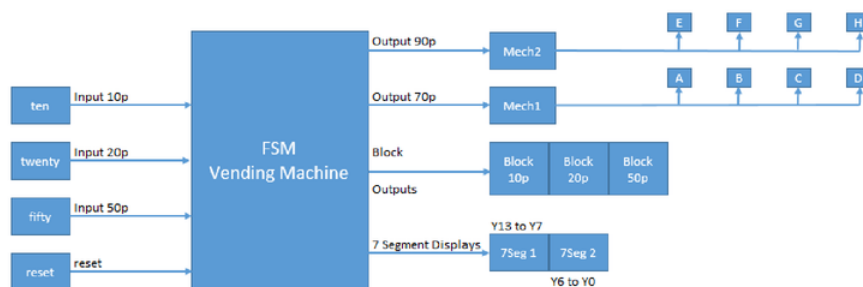


Figure 2 - Modified vending machine



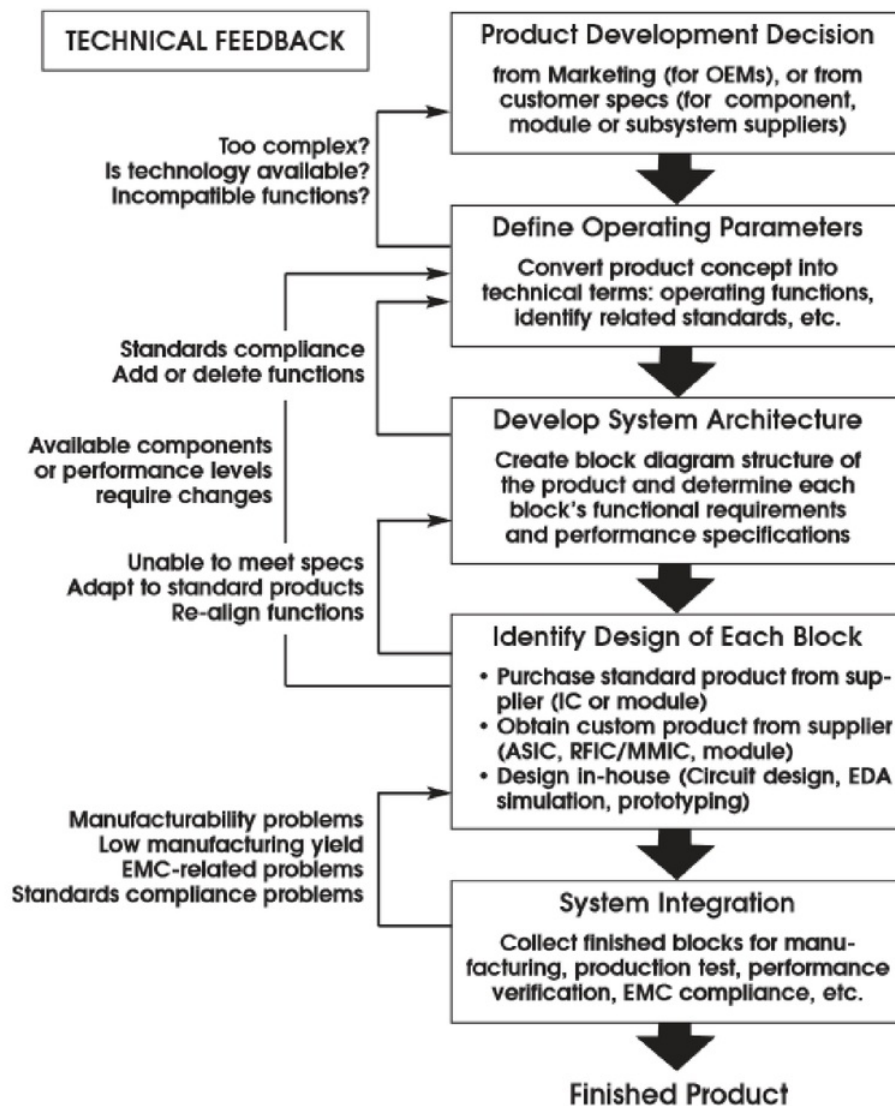


Figure 4 - Top down design flow chart

(Breed, 2008)

### 3. Design method, choices and calculations

The simple FSM vending machine described in the introduction has to be translated into a set of Boolean expressions. All inputs to the machine will be simulated with push buttons, 3 for the choice of coin and 1 for resetting the machine. The machine will have various outputs: 3 binary outputs for the number count which will be fed into a BCD-to-7seg decoder, 3 outputs to block each type of coin when the machine will no longer accept that type, and an output to say the goal of 70p has been reached.

#### State diagram

Using the functional block diagram shown in the introduction, the state diagram shown in Figure 5 was developed. Internal signals for 'blocking' types of coin from being entered activate when the coin entered would bring the total past 70p. The 10p is blocked at S7, the 20p is blocked at S6 and the 50p is blocked at S3. A signal will also be used to show that the machine has received a total of 70p. As only 1 coin can be entered at once, and there are only 4 coins (including 0p) available, all inputs can be represented with a 2-bit binary number, which will be labelled J and F.

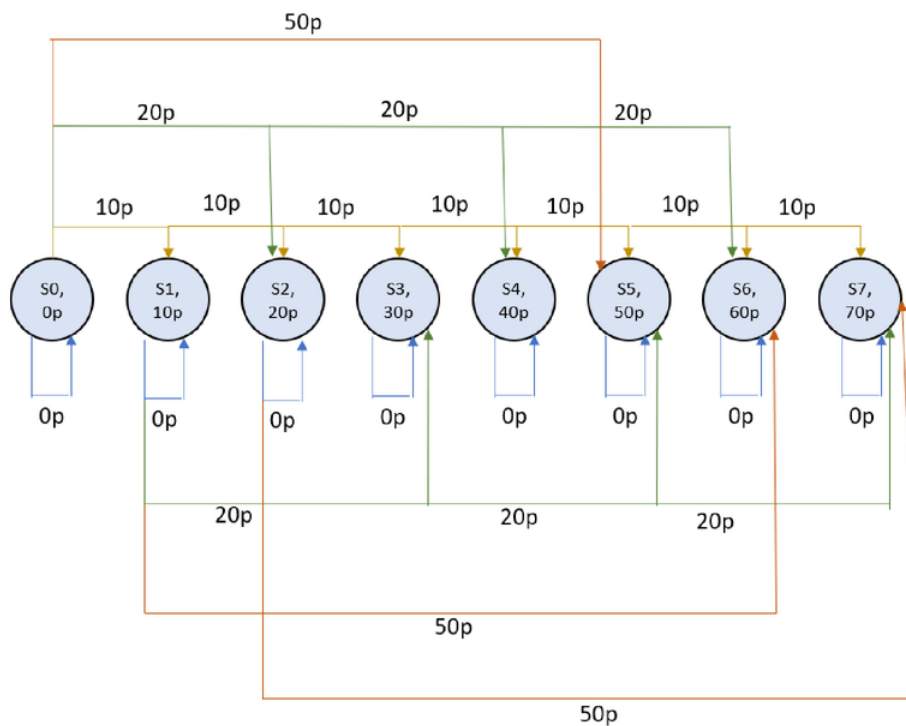


Figure 5- Vending machine state diagram

## J and F signals

The J and F signals have been chosen by designing a small truth table and Karnaugh maps. As only one coin can be entered at once, outputs from the truth table where 2 coins are input are going to be ignored.

| 50 | 20 | 10 | J | F |
|----|----|----|---|---|
| 0  | 0  | 0  | 0 | 0 |
| 0  | 0  | 1  | 0 | 1 |
| 0  | 1  | 0  | 1 | 0 |
| 0  | 1  | 1  | x | x |
| 1  | 0  | 0  | 1 | 1 |
| 1  | 0  | 1  | x | x |
| 1  | 1  | 0  | x | x |
| 1  | 1  | 1  | x | x |

Figure 6 - J and F truth table

The table in Figure 6 was used in the main truth table to represent all possible coin inputs using only 2 binary bits.

## Truth table

The state diagram was used to develop the truth table in Figure 7. The 70P output is only active once the machine has received 70p, the B10 B20 and B50 outputs are used as a signal that the corresponding coin has been blocked.

| Current state |          | Next state |       |       |       | Op                  | 10p                 | 20p                 | 50p                 | Output |     |     |     |
|---------------|----------|------------|-------|-------|-------|---------------------|---------------------|---------------------|---------------------|--------|-----|-----|-----|
|               |          | Op         | 10p   | 20p   | 50p   | JF=00               | JF=01               | JF=10               | JF=11               |        |     |     |     |
|               | Qa Qb Qc | JF=00      | JF=01 | JF=10 | JF=11 | $Q_a^+ Q_b^+ Q_c^+$ | $Q_a^+ Q_b^+ Q_c^+$ | $Q_a^+ Q_b^+ Q_c^+$ | $Q_a^+ Q_b^+ Q_c^+$ | B10    | B20 | B50 | 70p |
| 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   | 1   | 0   |
| S4            | 100      | S4         | S5    | S6    | S7    | 100                 | 101                 | 110                 | 111                 | 0      | 0   | 1   | 0   |
| S5            | 101      | S5         | S6    | S7    | S7    | 101                 | 110                 | 111                 | 111                 | 0      | 0   | 1   | 0   |
| S6            | 110      | S6         | S7    | S7    | S7    | 110                 | 111                 | 111                 | 111                 | 0      | 1   | 1   | 0   |
| S7            | 111      | S7         | S7    | S7    | S7    | 111                 | 111                 | 111                 | 111                 | 1      | 1   | 1   | 1   |

Figure 7 - Vending machine truth table

## Karnaugh map

To achieve simplified Boolean expressions, the truth table has been used to create a series of Karnaugh maps which will result in unnecessary or redundant Boolean expressions removed.

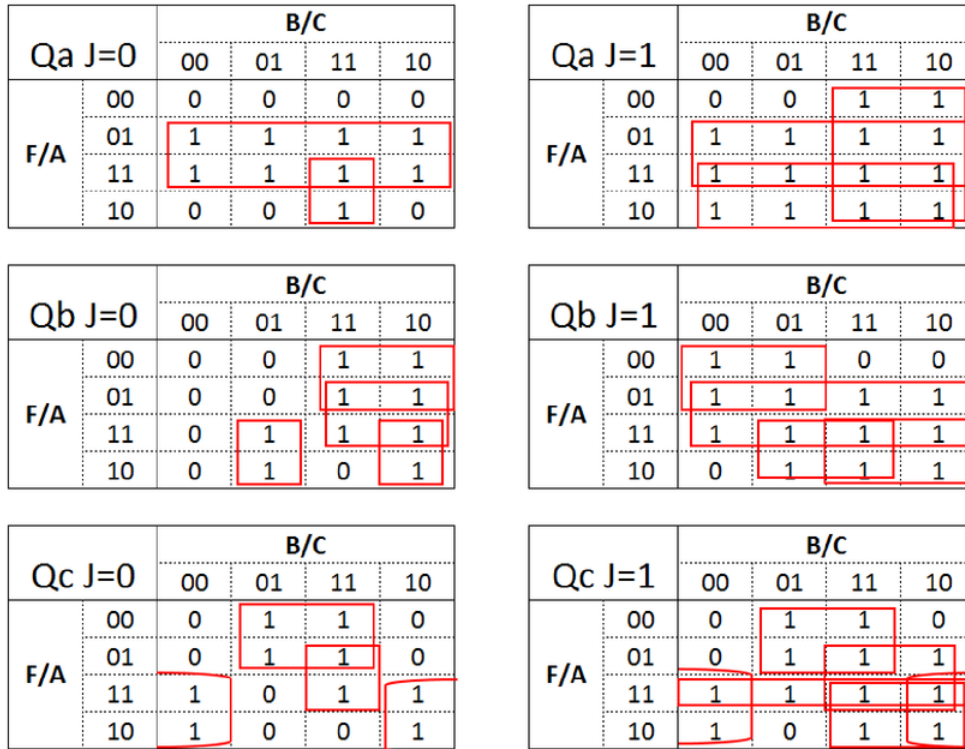


Figure 8- Karnaugh Maps

The final condensed Boolean expression derived from the Karnaugh maps in Figure 8 are:

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

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

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

### Multisim circuit

The Boolean expression derived from the Karnaugh map was converted into a Multisim logic circuit. Shown in Figure 9. To check the validity of the design, it has to be fully tested to ensure it functions as per the initial specification. Please see section 4 page 12 in the testing section of the report.

The design passed all given tests and functions as specified in the initial spec. The design was then rebuilt in Multisim PLD mode which allows the design to be downloaded onto an FPGA. Multisim FPGA mode only allows components that are compatible with the FPGA chip to be placed; simple logic gates, flip-flops, BCD-to-7seg etc. The seven segment displays are not available in FPGA mode which is why the initial design was tested in detail before attempting the FPGA design. The only way to test the FPGA using Multisim is to download it to the board, which is a lengthy process compared to adding 7segs and simulating in normal mode. The downside to Multisim FPGA mode is any design created in normal mode CANNOT be copy-and-pasted into PLD mode and has to be rebuilt from scratch. This adds even more time to this quite time-consuming way of programming the FPGA.

Figure 11 shows a simple circuit that divides the standard clock frequency. The circuit consists of a series of flip-flops, with each successive flip-flop dividing the clock in half. There are various ways that the clock frequency can be reduced, this is just one simple method. Starting with a 64Hz clock and using 6 flip-flops effectively divides the input clock frequency in half 6 times.  $64/2 = 32/2 = 16/2 = 8/2 = 4/2 = 2/2 = 1$ . This shows that the circuit reduces the frequency from 64Hz to 1Hz.

Viewing the multiplexing circuit in Figure 12 shows that only 2 anodes are being cycled between. This is for the tens and units for the coins entered into the machine. For the purposes of downloading to the Nexys 4 board, the P is not being displayed. To implement a third digit to multiplex would take a re-design of the multiplexing circuit and would majorly increase its complexity. This is another drawback to using Multisim for FPGA design.



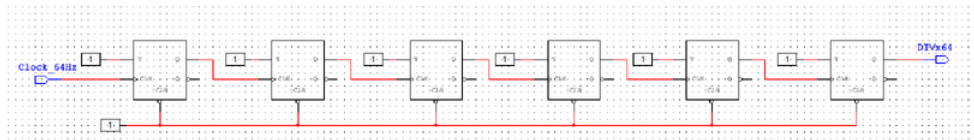


Figure 11 - Clock divider

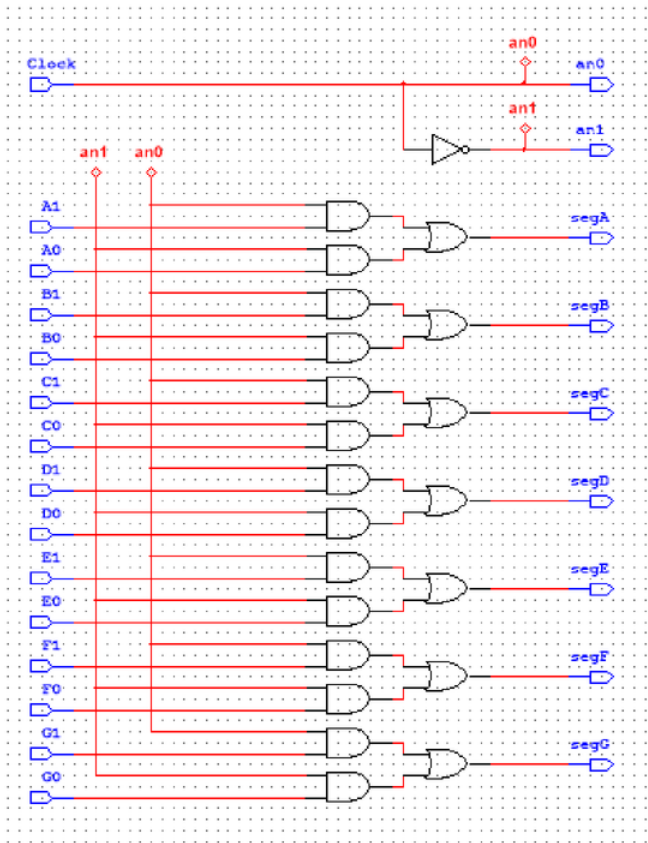


Figure 12 - Multiplexer

## 4. Testing and improving the design

### Multisim testing

To check the validity and functionality of the Boolean expressions that were developed from the Karnaugh map, the entire circuit was simulated in the normal simulation mode on Multisim. The outputs from each of the flip-flops are fed into a BCD-to-7seg decoder which is responsible for controlling the 'tens' bit. The 0 in the 'units' column and the 'P' are static and will not change, so the corresponding 7seg code has been hard wired onto the 7segs's pins. Each input was tested, and other features of the FSM were tested to check it adheres to the initial specification, see Figure 13.

The Boolean expression derived meets all the requirements set out in the introduction and was taken onto the next step. The next step was to develop a circuit in FPGA mode that can be downloaded to the Nexys 4 board, with the on-board buttons and 7-segs used as inputs/outputs respectively.

| Action                                        | Expected outcome                           | Pass/fail | Figure    |
|-----------------------------------------------|--------------------------------------------|-----------|-----------|
| Enter 10p                                     | 10p displayed                              | Pass      | Figure 14 |
| Enter 20p                                     | 20p displayed                              | Pass      | Figure 15 |
| Enter 50p                                     | 50p displayed                              | Pass      | Figure 16 |
| Enter coin when 70p has been reached          | Counter stays at 70p                       | Pass      | Figure 17 |
| Try entering each coin when counter is at 70p | Counter stays at 70p, blocks input of coin | Pass      | Figure 17 |
| Press Reset button                            | Counter returns to 0p (FSM is reset)       | Pass      | Figure 18 |

Figure 13 - Boolean expression test table

## Boolean expression testing evidence

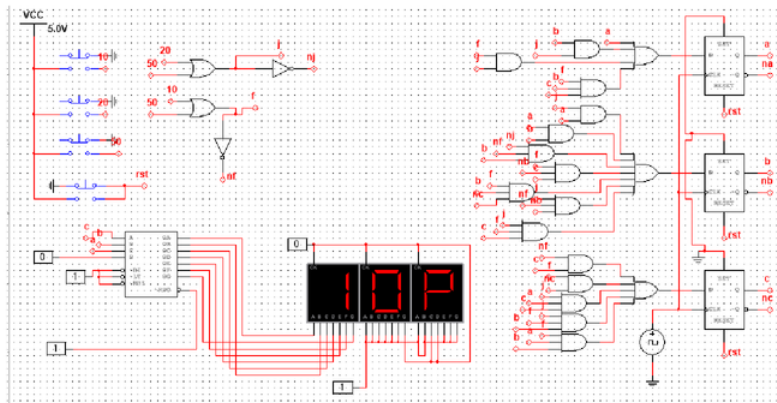


Figure 14- Boolean testing 10p

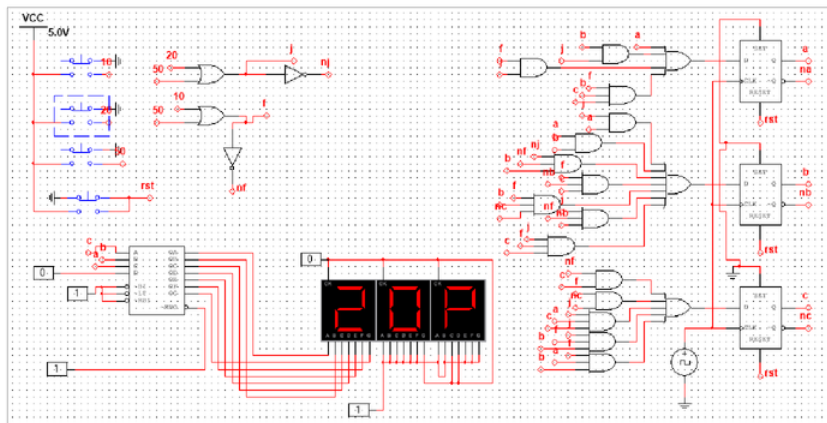


Figure 15 - Boolean testing 20p

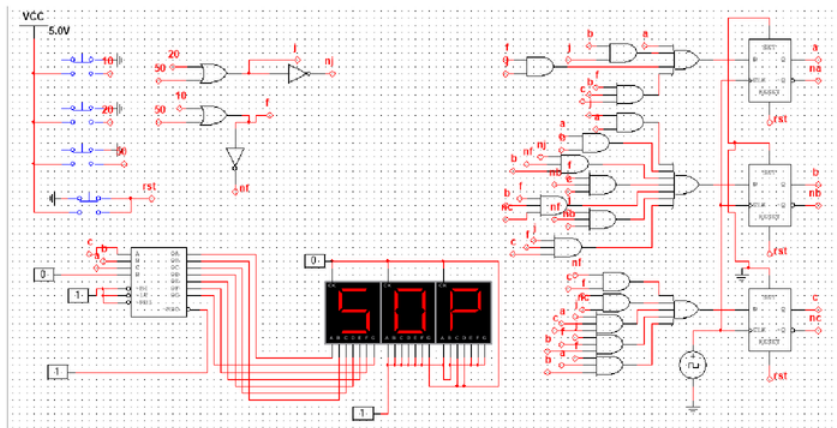


Figure 16 - Boolean testing 50p

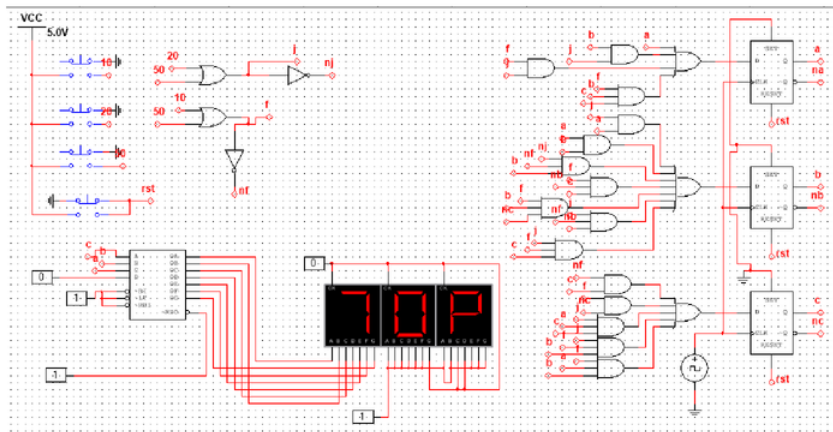


Figure 17 - Boolean testing 70p and coin blocking

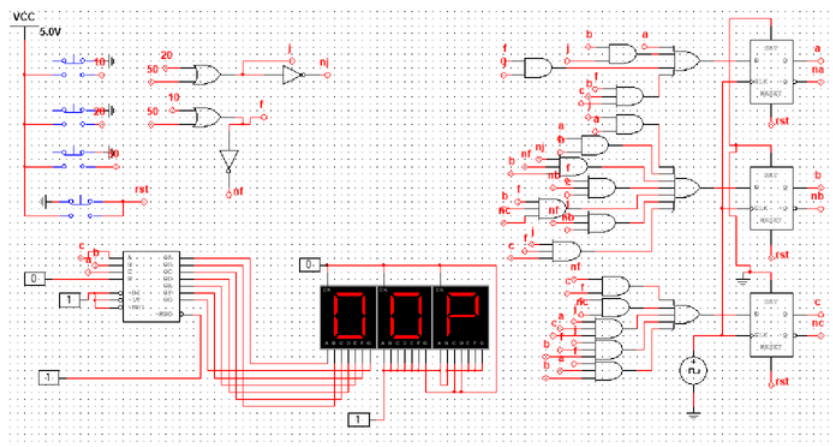


Figure 18 - Boolean testing reset

## Multisim FPGA testing

The following screenshots in Figure 19 show the Multisim FPGA design being tested on the Nexys 4 board. The 10, 20 and 50P coins have been tested separately, with the board being reset to 0 after each coin has been entered. The design worked as intended with the screenshots showing each coin being displayed on the 7seg displays. As mentioned, the 'P' symbol after the number was not implemented in the design to be downloaded to the board.

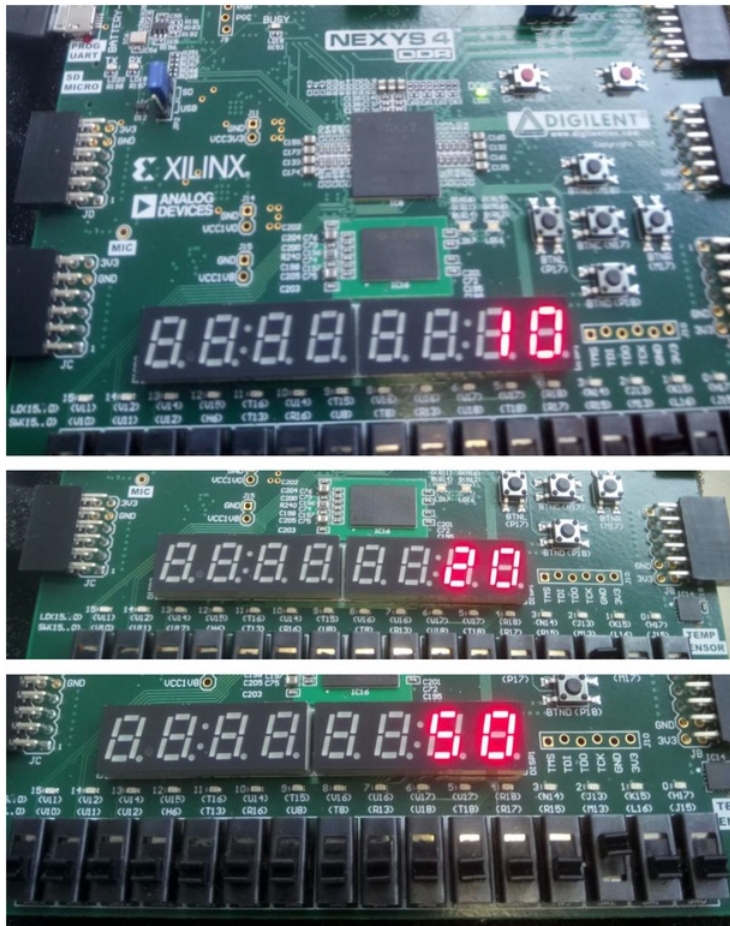


Figure 19- Nexys 4 10, 20 and 50p testing

## VHDL testing

VHDL testing with all 10p. Output 'seventy' comes on once 70P has been entered into the machine. Each type of coin that would make the total exceed 70P is blocked, as shown in Figure 20. The stimulus code used to test the program is also shown.

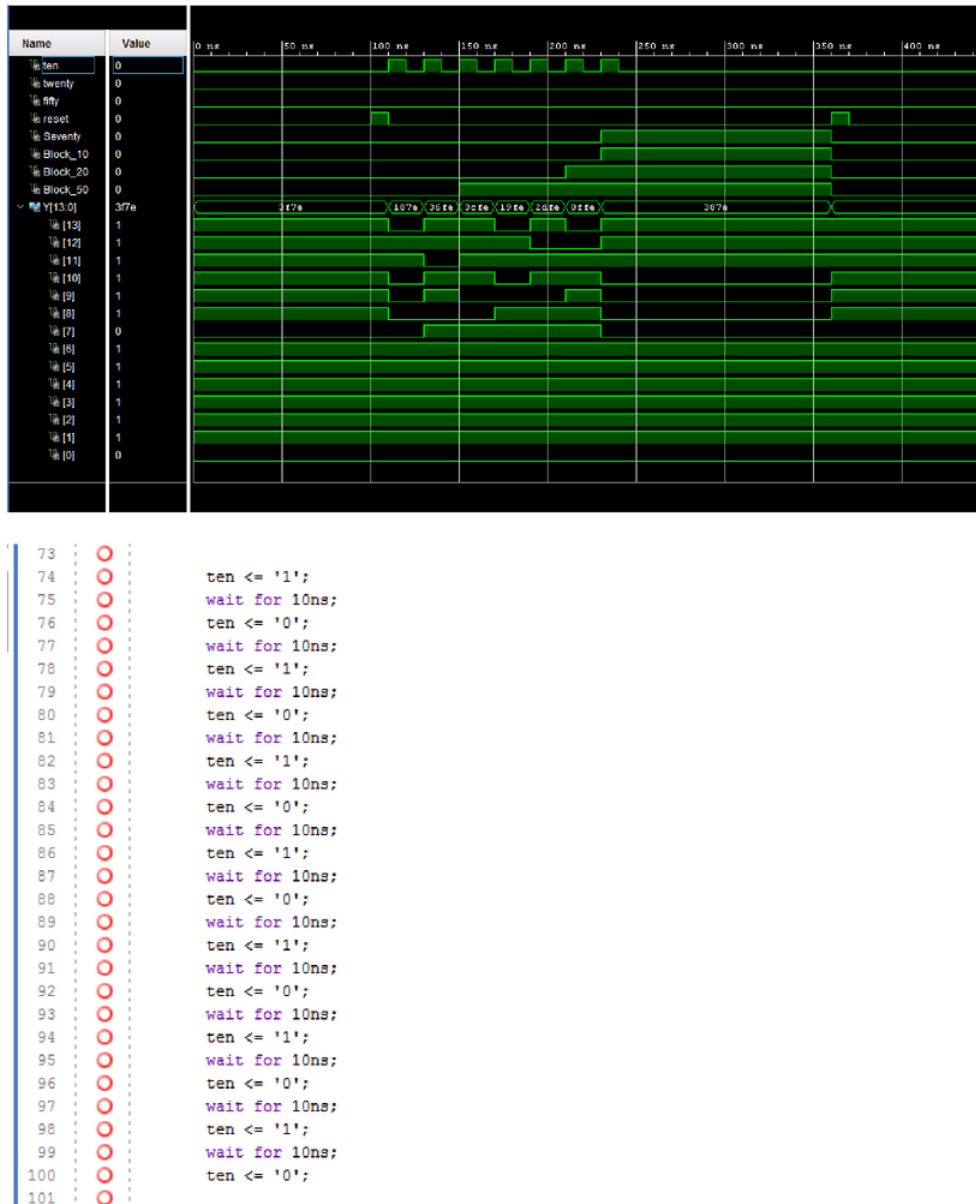


Figure 20 - VHDL test 1

Test 2 for the VHDL code is shown in Figure 21. Testing with 50p's, if 50p is put in when it is blocked, it will refund the coin and not count towards the total. Output 'seventy' again comes on once 70P has been entered into the machine.

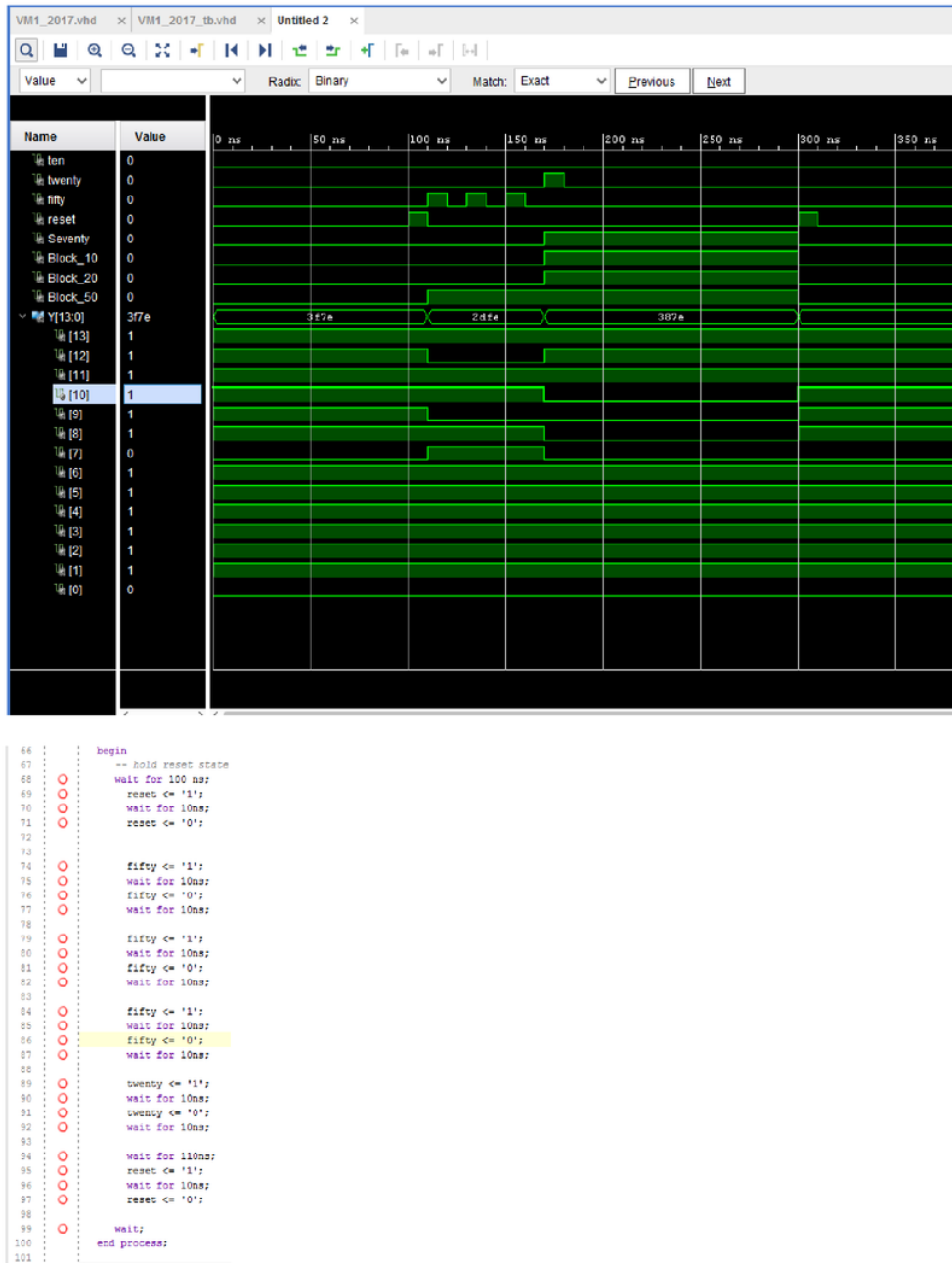
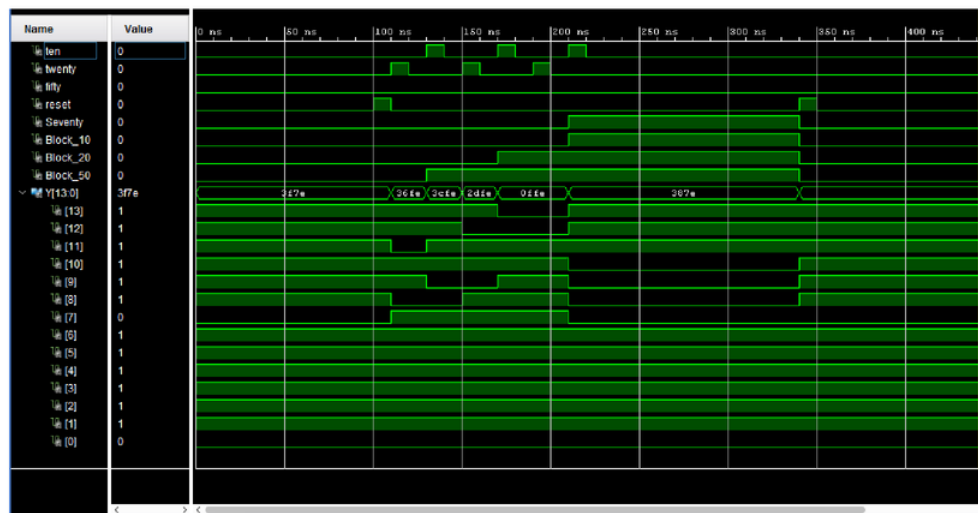


Figure 21 - VHDL test 2

To meet the first half of the initial specification, the VHDL design was downloaded to the Nexys 4 board. The test will be similar to the test carried out on the Multisim circuit, ensuring the 7 seg displays work correctly on the board.



```

74      ○          twenty <= '1';
75      ○          wait for 10ns;
76      ○          twenty <= '0';
77      ○          wait for 10ns;
78
79      ○          ten <= '1';
80      ○          wait for 10ns;
81      ○          ten <= '0';
82      ○          wait for 10ns;
83
84      ○          twenty <= '1';
85      ○          wait for 10ns;
86      ○          twenty <= '0';
87      ○          wait for 10ns;
88
89      ○          ten <= '1';
90      ○          wait for 10ns;
91      ○          ten <= '0';
92      ○          wait for 10ns;
93
94      ○          twenty <= '1';
95      ○          wait for 10ns;
96      ○          twenty <= '0';
97      ○          wait for 10ns;
98
99      ○          ten <= '1';
100     ○          wait for 10ns;
101     ○          ten <= '0';
102     ○          wait for 10ns;
103

```

18

### VHDL Nexys 4 testing

As with the Multisim test, the VHDL design was downloaded to the Nexys 4 board and each different coin was entered into the machine (10p, 20p, 50p) with a reset after each coin.

Figure 23 shows the results from the tests. Each coin value displays correctly on the LCD screen. As discussed earlier, the 'P' symbol has not yet been programmed into the VHDL code.

The initial VHDL functions as expected and meets the first part of the initial specification. The VHDL code will be modified in the next section to be able to meet the second part of the specification.



Figure 23 - VHDL first design Nexys 4 board display

### Further improving VHDL design

To meet the initial specification of the assignment, the VHDL design has to be modified to include 2 possible outputs, 70p and 90p. Once either one of the set point prices has been reached, an output labelled 'Release' will activate which corresponds to the machine outputting a product to a customer. There is a small selection of inputs which allows the customer to select a product of that particular price range, the inputs used are A, B, C, D, E, F, G and H.

A to D are for the 70p output, E to H are for the 70P output. Although not currently implemented, adding the 'P' letter to the 7 segment displays would only require adding 7 more bits to the Y signal array. As the P will never change, the values used in the signal array will be static in the exact same way the 0 is static and constantly displayed.

Test 1 in Figure 24 shows 50p, 10p, 10p, 10p and 10p being entered into the machine giving a total of 90p, which is confirmed by the 'ninety' output activating. Inputs A and C have been pressed and nothing is released. This is expected as those outputs are for 70p products and the initial specification says only correct amounts are recognised and no change is given. The F input is then pressed and the release output is activated.

Test 2 in Figure 25 shows 20p, 20p, 20p and 10p being entered and the 'seventy' signal activates as expected. Inputs E and G are pressed which does nothing, as they are for the 90p mech. The 'release' signal activates once the B input is pressed which is for the 70P mech.

Test 3 in Figure 26 shows the same as Test 2, except with the 7-segment display signal array being expanded. The bits can be seen changing in response to each coin being entered.

The full modified code for the VHDL file and the associated testbench file are in the appendix of the report.

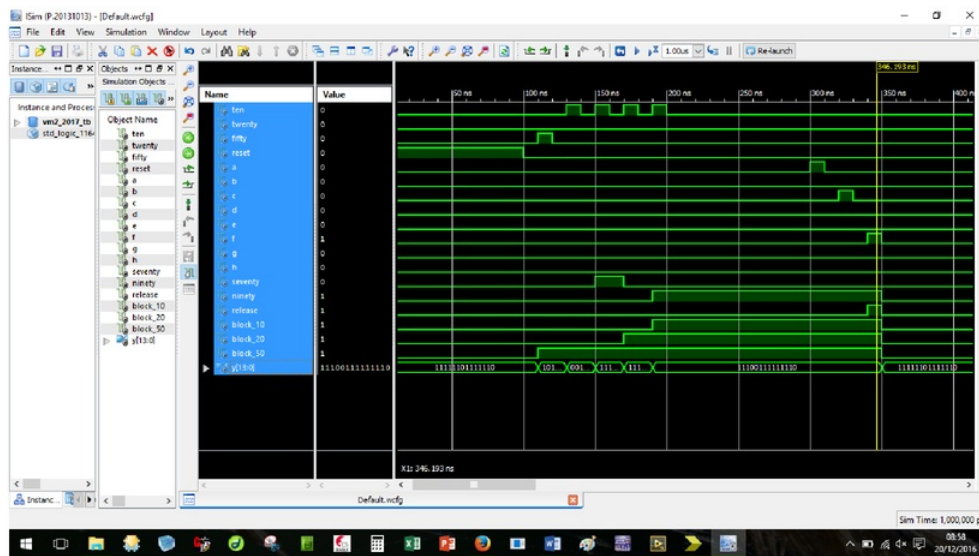


Figure 24 - Modified VHDL test 1

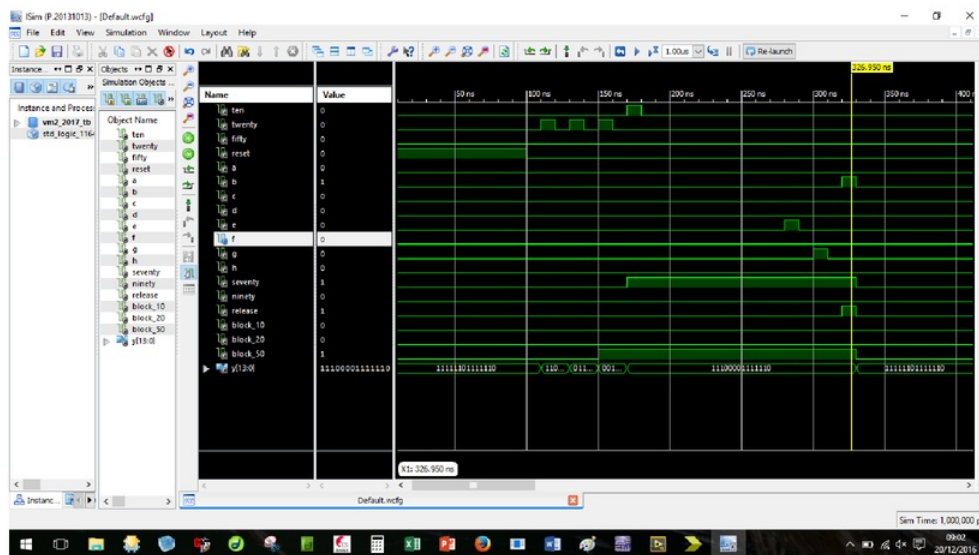


Figure 25- Modified VHDL test 2



## 5. Conclusion

Overall the assignment was a success. The circuit described in the specification was produced using 2 different methods, VHDL and Multisim PLD. The two methods happen to take different design approaches, top-down and bottom-up as described in the research section. There is a major difference in how efficient each software is at creating a program for the FPGA:

Multisim is entirely visual with the entire circuit requiring only logic gates and no code. This is incredibly useful for someone trying to learn this type of programming for the first time but some major drawbacks became apparent during this assignment. The main drawback to the Multisim PLD design is how difficult it is to modify once the circuit has been designed. The entire process of state diagram, truth table, Karnaugh maps and Boolean expressions has to be redone if a change to the circuitry was required, such as incorporating a 90P mech and various buttons.

The Vivadoo VHDL on the other hand has a very steep learning curve for beginners, as coding is required and the structure of the VHDL program is different than conventional C programming for example. However, VHDL benefits intermediate users by allowing every single aspect of the program to be built entirely from scratch. Instead of using premade logic gates on Multisim one can simply program the exact logic they want using VHDL. The VHDL program also benefits from being simple to modify, as has been proved in the assignment.

Another key difference between the software packages are how complex a circuit can get: As you keep adding more circuitry to the Multisim design, it begins to get very large. As multisim is very visual, this leads to an incredibly large schematic with many wires and on/off page connectors. These can sometimes be tricky to fault find and gets more difficult the larger the schematic.

Vivadoo on the other hand benefits from being a (largely) text-based design, which is laid out in a way to make it much easier to follow what is going on in a complex program once the general structure of VHDL programs is known. Another benefit of using Vivadoo is that designs can be made into IP blocks. Once an IP block is created from a user's code, it is converted into a component with inputs and outputs associated with the VHDL code inside. This means many largely complicated designs can be split up into separate IP blocks, and then connected as one would normally connect the pins of IC's. If for example the vending machine design had a lot more features, it may have made sense to split it up into IP blocks, for example: An IP block could be made to detect the coins. This could then connect to an IP that release a product based on a user's input. Although this method is not practical for a system this simple, the example was just to show how this method could be used.

The background research shed light on design methods that were employed in this assignment:

The top-down approach was the method that was used to develop the Multisim circuit, using pre-built components with limited customisability to achieve the initial specification. The bottom-up design approach that was researched happens to be the method used with the VHDL design. The fully customised components that were designed specifically for the purpose of the initial specification. The pros and cons of both methods have been proven throughout the report.

The bottom-up approach in this particular assignment proved to be the most efficient to complete the set tasks. The Top-down approach would have required lots of time to re-design the FSM to add some features to be able to add a second mech to the vending machine. The top-down approach on the other hand was simple to modify by adding bits of code. This was made possible because each component was designed from scratch to meet the specification, so small modifications to the components was enough to add functionality to the design.

## 6. Reference list and bibliography

### References

Breed, G., 2008. *Top-Down Design: An Introduction for New Engineers*. [Online] Available at: [https://www.highfrequencyelectronics.com/Jan08/HFE0108\\_Tutorial.pdf](https://www.highfrequencyelectronics.com/Jan08/HFE0108_Tutorial.pdf) [Accessed 14 2018].

Bruno Wacogne, S. P. A. G. K. C. L. P., 2017. Medical Devices Development: The Bottom-Up or The Top-Down Approach?. *International Journal of Biosensors & Bioelectronics*, 3(5), pp. 1-7.

Gajski, D. A. S. G. A. S. G., 2009. *Embedded System Design*. 1st ed. s.l.:Springer US.

Indovina, C. B. G. G. M., 2014. *A TOP-DOWN APPROACH TO IC DESIGN*. [Online] Available at: [www.indovina.us/~mai/a\\_top\\_down\\_approach\\_to\\_ic\\_design.pdf](http://www.indovina.us/~mai/a_top_down_approach_to_ic_design.pdf) [Accessed 5 04 2018].

Loew, M., 2013. *A Working Definition of Systems Engineering*. [Online] Available at: <https://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/6576/Bottoms-up-or-is-the-Top-down-A-Working-Definition-of-Systems-Engineering.aspx> [Accessed 24 2018].

## 7. Appendix

### VHDL code

```
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;

elseif (ten='1') then

n_s <= S1;

elseif (twenty='1') then

n_s <= S2;

elseif (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;

elseif (ten='1') then

n_s <= S2;

elseif (twenty='1') then

n_s <= S3;

elseif (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;

elseif (ten='1') then

n_s <= S3;
```

```

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

```

## Testbench

```
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';
    end process;

```

```
        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;
```

# Ass2 EDES final

## 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

GRADING FORM: CSG-RUBIC

**MICHAEL LEWIS**

**87**

## RESEARCH-INFORMED LITERATURE



There was good evidence of further reading and research from credible and reliable sources. Your interpretation and understanding of the additional research was good and demonstrated a very good underlying appreciation of the subject. Your references were appropriate to the task. 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. You could however have explained the VHDL code in more detail.

**25**

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

**18**

## PRACTICAL APPLICATION AND DEP.



Your application of practical skills for this task was generally excellent. You used the resources available in MultiSim and Vivado 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.

**27**

## 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