STATEMENT OF ACADEMIC INTEGRITY

I confirm and declare that this report and the assignment work is entirely the product of my own efforts and I have not used or presented the work of others herein.

Jee Ken Chung

Steven Webster

Fraser Rigby

SEMESTER 2 ASSIGNMENT

EE270 - DIGITAL ELECTRONIC SYSTEMS

Contents

Abstract

This assignment created VHDL designs to implement onto a Basys-3 board. This design aimed to display the birthdays of each group member on the board's four 7-segment displays using the format (DDMM). Only one birthday was displayed at any time and the birthday that is displayed was determined by the current inputs of the board.

1 Introduction

1.1 The Design Task

On a Basys-3 FPGA board, the birthdays of the three group members were to be displayed on four 7-segment displays. The display of each birthday must follow the format of (DDMM, where D was a single digit of the birthday date and M was a single digit of the birthday month). Only one birthday is to be displayed at any given time. The current birthday on display was determined by 3 different input permutations. One birthday would correspond to one permutation

In addition to displaying a birthday, each of the three permutations will light up a unique LED sequence on the board.

1.2 Assumptions Made and Design Decisions

It was decided that the switches of the basys-3 board would be used as inputs (instead of its pushbuttons). This was to avoid the issue of "bouncing", which pushbuttons commonly experience. This would serve to simplify the VHDL code as it would not need to implement any debouncing features.

There were three team-members and so three different input permutations were required. If each team-member had one switch each (a HIGH on this switch would indicate that their information was to be displayed), then three switches would be required. An issue would occur when more than one switch was HIGH: whose information should be displayed? After clarifications from Dr Crockett [1], it was revealed that the design did not need to handle such unexpected inputs, but it could be done as an optional extension.

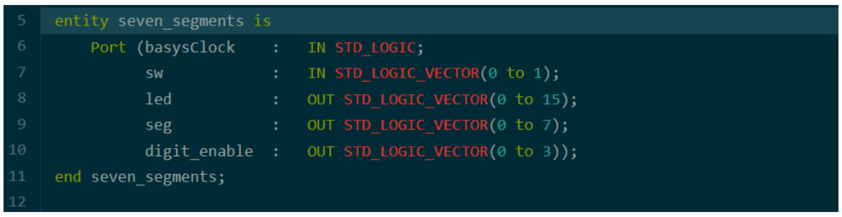
In the final code, only two switches were used. Each switch would act as one bit of a 2-bit number and allow for 4 different input permutations. Each team-member would be assigned exactly one permutation. For the fourth and final permutation, the error message would be displayed (a special display given to the LEDs and the 7-segment displays). In doing this, the code was able to deal with these unexpected inputs.

Furthermore, by treating the switches as bits to a binary number, they could be expressed as a 2-bit `STD_LOGIC_VECTOR`. A straight-forward conversion could then be made to the `UNSIGNED` data type and then an `INTEGER` data type. This `INTEGER` could then be used as an index to the arrays of the code (which shall be discussed later).

2 Top-Down Discussion of the Design

2.1 Overall Functionality

The design's ports/interface was seen in the entity declaration shown in Figure 1.


```
entity seven_segments is
    Port (basysClock    :   IN STD_LOGIC;
          sw            :   IN STD_LOGIC_VECTOR(0 to 1);
          led           :   OUT STD_LOGIC_VECTOR(0 to 15);
          seg           :   OUT STD_LOGIC_VECTOR(0 to 7);
          digit_enable  :   OUT STD_LOGIC_VECTOR(0 to 3));
end seven_segments;
```
Figure 1: Design's entity declaration

The ports had the following purposes:

- `BASYSCLOCK` – A clock was required as some notion of time-keeping was required in the design; each pattern of the sequence would have to stay on for some specified amount of time.

    - The VHDL code used clock dividers in order to achieve the appropriate clock speeds. However, these dividers assumed that the clock given into the port was that of the basys-3 clock (100 MHz [2]). Any other frequency used for this port would have affected the internal timings of the design
- `SW` – This was connected to two different inputs on the basys-3 board. The port's value would determine which birthday and sequence would currently be on display
- `LED` – This was connected to the 16 LEDs on the basys-3 board. It would be used to display a pattern of the LED sequence
    - The current sequence that was on display was determined by those `SW` inputs
- `SEG` – Controlled the 8 different segments available on the 7-segment display
    - It would show the segment values required to display one digit and was used with `DIGIT_ENABLE` to show all four digits of a birthday
- `DIGIT_ENABLE` – A 4-bit port that was used to enable or disable each of the four 7-segment displays.

2.2 Creating Data Types to Represent a Birthday and an LED Sequence

It was conceived that the design would make use of arrays of arrays. For example, there were 16 LEDs that were used to show the LED sequence. Hence, each pattern of the sequence could be represented as a `STD_LOGIC_VECTOR` of 16 bits. By creating an array of these, it would be possible to encapsulate the entire sequence to be displayed. To do this, a new array type was created called `LED_DISPLAY`. The size of `LED_DISPLAY` was constrained so that it would hold exactly 16 of these `STD_LOGIC_VECTORS`. This meant that 16 patterns were in each sequence.

For the 7-segment display, all segments of one display were represented through another `STD_LOGIC_VECTOR` type of 8 bits. This was enough to describe the values that all of the segments. Although, a convention was required to map each index of this array to a single segment of the display. This mapping was shown in Figure 2.

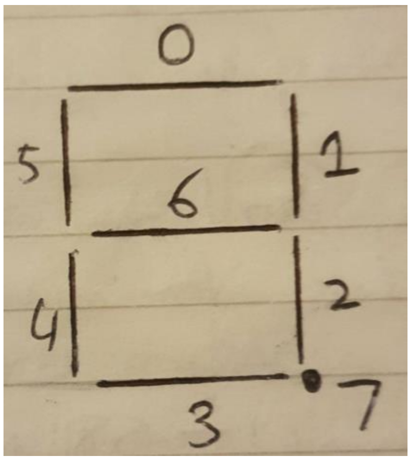


Figure 2: Mapping of the STD_LOGIC_VECTORs' indices to segments [3]

For example, to show the number 1, the `STD_LOGIC_VECTOR` would be "10011111"[1]. Like the LED sequence, an array was made to hold those arrays. This array was called `SEGMENTS_DISPLAY` and held four of these arrays, enough to store the 4-digit birthday date.

[1] A LOW value was required to activate a segment as these signals were the cathodes to each of the segments.

2.3 Working with the LED_DISPLAY and SEGMENTS_DISPLAY Data Types on the Basys-3 Board

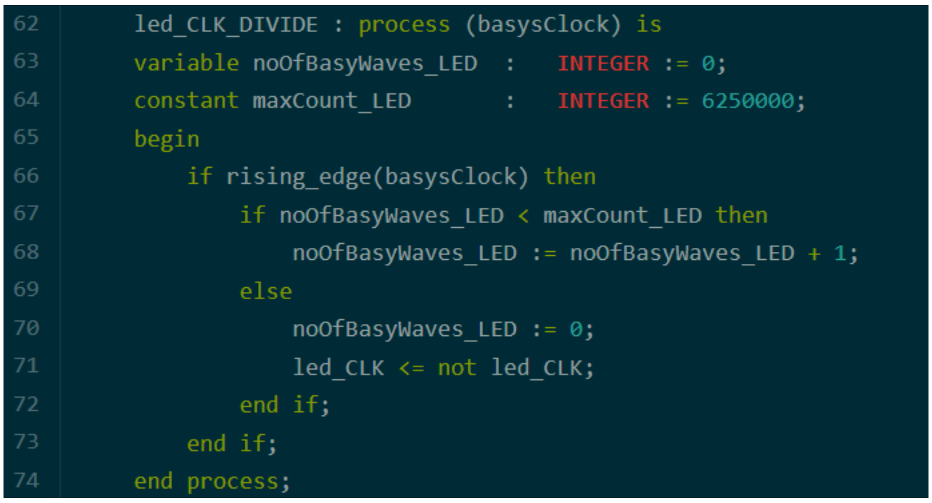
2.3.1 LED_DISPLAY and the Clock Divider

2.3.1.1 Clock Divider

Each pattern of the LED sequence should be on for a specified amount of time. This required some measurement of time. Therefore, a clock signal was required to manage this; the next pattern would be moved onto when a rising edge was read in. The Basys-3 board had a 100MHz clock that could be used in the VHDL design [2]. However, Equation (1) shows that the period for which each pattern would be shown was 0.01 µs. Such a small amount of time would not have been perceivable to the human eye. Instead, it would appear as if all patterns were on simultaneously, which was not desired.

$$\mathrm{T_{basys}} = \frac{1}{\mathrm{f_{basys}}} = \frac{1}{100\ \mathrm{Mhz}} = 0.01\ \mu\mathrm{s} \qquad (1)$$

Therefore, a clock divider was required in the VHDL design to make the sequence visible. Previous work with LED sequences showed that a rate of 8 Hz was perceivable [3]. Thus, the design code included the VHDL process of Figure 3.



```
62      led_CLK_DIVIDE : process (basysClock) is
63      variable noOfBasyWaves_LED  :   INTEGER := 0;
64      constant maxCount_LED       :   INTEGER := 6250000;
65      begin
66          if rising_edge(basysClock) then
67              if noOfBasyWaves_LED < maxCount_LED then
68                  noOfBasyWaves_LED := noOfBasyWaves_LED + 1;
69              else
70                  noOfBasyWaves_LED := 0;
71                  led_CLK <= not led_CLK;
72              end if;
73          end if;
74      end process;
```

Figure 3: Clock division used for the LED sequence

The `noOfBasyWaves_LED` integer was initialised to 0 in Line 63. Line 66 would then check if the basys-3 clock had a rising edge (i.e. whether one complete wave had passed). If it had, then `noOfBasyWaves_LED` would increment in Line 68. Thus, it counted how many clock waves had passed. However, note that it does not always increment; it only increments when `noOfBasyWaves_LED` was less than `maxCount_LED`. When `noOfBasyWaves` reached this value, lines 70 and 71 were executed instead. This reset the `noOfBasyWaves_LED` variable and inverted `led_CLK`[1] so that it had one edge created at this time. It follows, therefore, that the relationship of Equation (2) held.

$$\text{Time taken for one led\_CLK edge to be created} = \mathrm{T_{basys}} \times \text{maxCount\_LED} \qquad (2)$$

The equation made it clear that `maxCount_LED` was central in determining what the period of `led_CLK` would be. Although, it was important to realise that since only one edge was created, only half of the wave had been generated. A full wave would only pass if two edges had passed. Therefore, to obtain the period of the `led_CLK`, Equations (3) and (4) would be required.

[1] This was the clock signal that was eventually used to control the LED sequence. Each rising edge of it would move the sequence onto its next pattern.

$$\mathrm{T_{led\_CLK}} = 2 \times \text{Time taken for one led\_CLK edge to be created} \tag{3}$$

$$\mathrm{T_{led\_CLK}} = \frac{2\mathrm{T_{basys}}}{\mathrm{maxCount\_LED}} \tag{4}$$

For this project, it was more useful to deal with frequencies than periods. The equations should also be used to determine what MAXCOUNT_LED should be in order to gain the desired clock frequency. Hence, the previous equations were manipulated to give Equation (5).

$$\mathrm{maxCount\_LED} = \frac{\mathrm{f_{basys}}}{2\mathrm{f_{led\_CLK}}} \tag{5}$$

Since 8 Hz was desired and the basys-3 clock was 100 MHz, MAXCOUNT_LED could be determined from Equation (6).

$$\mathrm{maxCount\_LED} = \frac{100\ \mathrm{MHz}}{2 \times 8\mathrm{Hz}} = 6\,250\,000 \tag{6}$$

As seen in Figure 3, this value was indeed used in line 64.

Whenever LED_CLK experiences a rising edge, the next pattern in the sequence should be displayed. In LED_DISPLAY, this was done by indexing into the next element to obtain the STD_LOGIC_VECTOR representing the next pattern to be displayed.

2.3.1.2 Tracking which Pattern to Display

A separate signal was required to keep track of which pattern was currently on display. This would be used to index into the LED_DISPLAY array.

As there were 16 elements in the LED_DISPLAY array, this signal would require 16 different distinct values. Therefore, it was decided that the signal would be a STD_LOGIC_VECTOR of 4 bits. In the final code, this signal was called LED_SEQUENCENUMBER.

2.3.2 SEGMENTS_DISPLAY

2.3.2.1 The Idiosyncrasies of the Basys-3's 7-Segment Displays

It was important to realise that the basys-3 board could not light up different digits on all 7-segment displays simultaneously. This was due to how its electronics was set up [2].

On all of its 4 displays, the segments had a common cathode[1] to the same segments of all the other displays. Hence (assuming that the anode to all the displays were appropriate), one would be forced to display the same digit on all displays. This would not have been appropriate for displaying the team's birthdays.

To display different digits, the anodes had to be managed as well. In contrast to the cathodes, each display received their own anode. This anode would be shared with all segments in that display. Thus, in order to display different digits on the displays, one had to set the cathode segments of the first display/digit, set that displays anode correctly, and disable all the other displays by setting their anodes appropriately. One would then have to very quickly turn off this first display (through its anode), turn on the second display (through its anode), and set the segment values so that the correct digit for this second display was shown. This would likewise occur for the third and fourth displays.

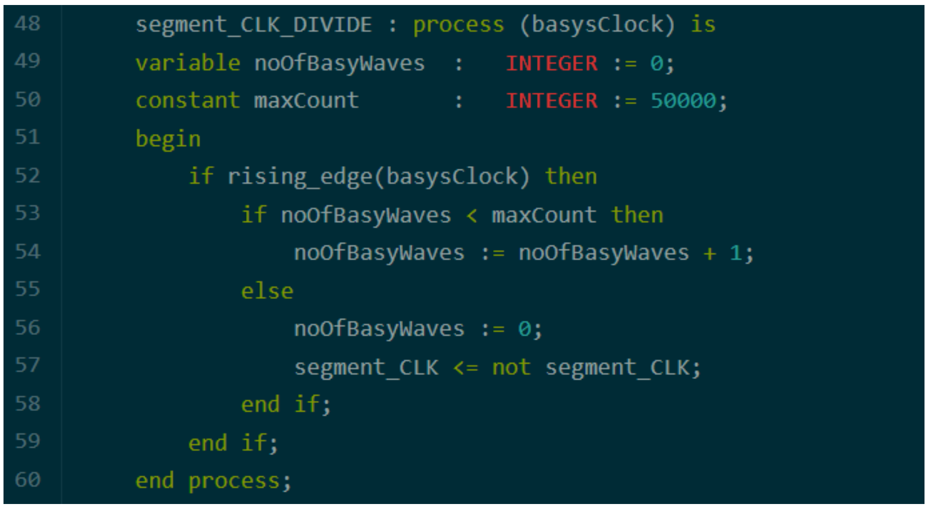
[1] The value of these cathodes was what was stored in the STD_LOGIC_VECTORS used for the 7-segment displays.

So long as this process occurred at a fast-enough rate, it would appear as if all displays were on at the same time. For all displays to "appear bright and continuously illuminated", the board's manual suggests that this occur at a rate of 1 kHz [2].

This desired frequency was much greater than that for the LED sequence. Consequently, a new clock signal had to be designed which had this frequency. A VHDL process was created to create this new clock. This process was the same[1] as that used to generate the LED sequences' frequency (*see* Figure 3). However, the value given to the constant `MAXCLOCK` was recalculated to consider the desired frequency of 1 kHz. Equation (5) was reused to give Equation (7).

$$\text{maxCount} = \frac{100 \text{ MHz}}{2 \times 1\text{kHz}} = 50\,000 \tag{7}$$

The final process used was that of Figure 4.



```vhdl
    segment_CLK_DIVIDE : process (basysClock) is
    variable noOfBasyWaves   :   INTEGER := 0;
    constant maxCount        :   INTEGER := 50000;
    begin
        if rising_edge(basysClock) then
            if noOfBasyWaves < maxCount then
                noOfBasyWaves := noOfBasyWaves + 1;
            else
                noOfBasyWaves := 0;
                segment_CLK <= not segment_CLK;
            end if;
        end if;
    end process;
```

Figure 4: Process used to create the clock for the 7-segment displays

Whenever a rising edge was in the `SEGMENT_CLK`, the following would occur:

- The anode of the current display would turn off
- The anode of the next display would turn on
- `SEGMENTS_DISPLAY` would index into the next element. This would give the segment values required for to display the correct digit on this next display

2.3.2.2 Tracking which Digit to Display

As there was 4 displays/digits, a `STD_LOGIC_VECTOR` of 2 bits was created. This would serve two purposes:

- It would serve as an index into the `SEGMENTS_DISPLAY` array
- At the same time, it would activate the correct anode for the display
    - For example, when the leftmost display/digit was to be activated, this `STD_LOGIC_VECTOR` would hold a value equivalent to 0 in decimal. Hence, it would index into the zeroth element of the `SEGMENTS_DISPLAY` array. At the same time, another part of the code would see its value of 0 and deactivate all anodes but the one belonging to that leftmost display.

This `STD_LOGIC_VECTOR` was a signal called `SEG_SEQUENCENUMBER`.

[1] Albeit, changes were made to the variable names to ensure clarity and to avoid confusion. This new clock was also created in its own, separate process.

2.4 Creating Greater Data Types to Encapsulate All Birthdays and All Sequences

Each instance of the data types discussed in Section 2.2 would only be enough to display the birthday and LED sequence of one person. Three instances were needed for the three people in this group.

To make the code cleaner, it was decided that an array holding all birthdays (i.e. all `SEGMENTS_DISPLAY` instances) would be made (called `TEAMS_BIRTHDAYS`). A second array would be made to hold all sequences (i.e. all `LED_DISPLAYS` instances). This array would be called `TEAMS_SEQUENCES`. These arrays would be the same size as each other. Furthermore, each team-member would be associated with one index of these arrays.

The rationale behind this association was to allow for code that was easier to use. In order to display a person's birthday or sequence, all that was required was the person's index and these arrays. The associations made were that of Table 1.

Table 1: The Index given to Each Group Member

Team Member	**Index Given**
Jee Ken Chung	0
Fraser Rigby	1
Steven Webster	2
ERROR DISPLAY[1]	3

The index that was currently in use would be determined by the inputs. This would allow the inputs to select a different person's information to be displayed, as per the design requirements. As there were two switches that were used (*see* Section *2.1*, p2), there was a total of $2^2 = 4$ input permutations. With the ERROR DISPLAY included in these arrays, the VHDL could be simplified so that the input permutation could act as an index into the array and access all elements (when converted to an integer data type). Hence, this was done to shorten the code.

2.5 Using All Those Arrays and Indices Together

Before a value could be assigned to the outputs, one of the four elements of the outer arrays (of `TEAMS_BIRTHDAYS` and `TEAMS_SEQUENCES`) had to be indexed into. The index that was used was an integer conversion of the 2-bit `STD_LOGIC_VECTOR` input of the `SW` port.

The elements of these outer arrays would each return another array. `TEAMS_SEQUENCES` would return `LED_DISPLAYS`, an array of 16 values that encapsulated one whole LED sequence[2]. A 4-bit number was also required to manage these LED sequences. This number would specify which pattern of the sequence would be shown. It would serve as an index into the `LED_DISPLAYS` array that was obtained from `TEAM_SEQUENCES`.

Note that the behaviour of this number was independent from the index into TEAM_SEQUENCES. Whereas the index into TEAM_SEQUENCES would be controlled by the input ports, this number would be controlled by a clock signal. It would increment on every rising edge of the clock for the LED sequence.

[1] An error display was also included in the array. This would have its own "birthday" and sequence associated with it. It would only appear if unexpected or erroneous inputs were given.

[2] This was due to the fact that each of those 16 elements were 16-bit `STD_LOGIC_VECTORS`. These `STD_LOGIC_VECTORS` gave the state that all LEDs should have in order to show one pattern of the sequence. Since 16 of these were stored in `LED_DISPLAYS`, 16 different patterns could be shown and hence, one whole sequence was encapsulated.

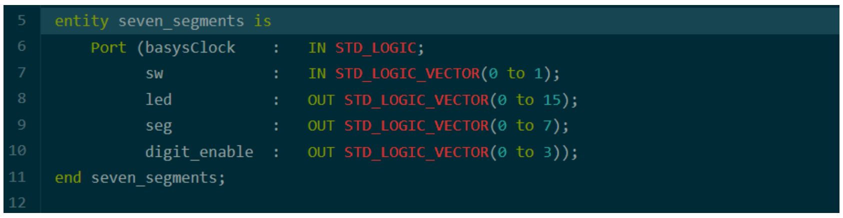
The same idea applied to `TEAMS_BIRTHDAYS`. The 2-bit input would be used as an index into this 4-element array. Indexing into this returned a `SEGMENTS_DISPLAY` array. This inner array contained the segment values for each of the 4 digits of the birthday. As only one digit could be shown at a time, another number was required to index that inner `SEGMENTS_DISPLAY` array. The index would change on every rising edge of the clock for the 7-segment displays. The index also had a further purpose in being used to enable or disable a display.

For example, if the leftmost display was to be shown, the index would be equivalent to 0, returning the zeroth segment values in the `SEGMENTS_DISPLAY` array. Another part of the code would read in this 0 and send an active signal to the leftmost display and an inactive signal to all the others. Once the index incremented to 1, the segment values of the second leftmost display would be returned. That other part of the code would see that the index was equivalent to 1, enable the second-leftmost display and disable all the others.

3 Implementation in VHDL Code

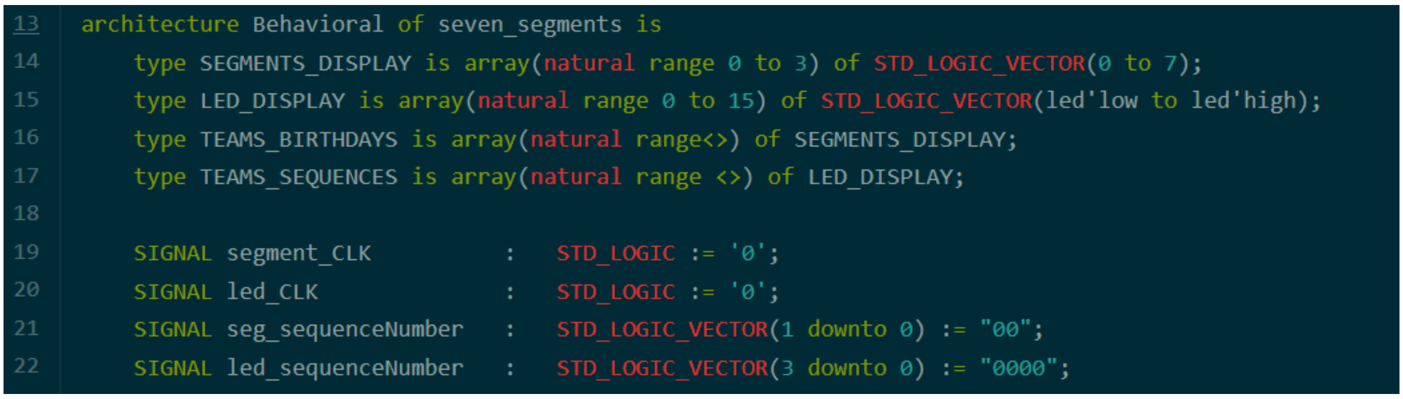
The full design code was made available as an appendix. This section shall go through each part of the code and say how it implements the ideas mentioned previously.

The entity declaration of Figure 6 implemented all the ports and interfaces mentioned in Overall Functionality, p2.



```
entity seven_segments is
    Port (basysClock    :   IN STD_LOGIC;
          sw            :   IN STD_LOGIC_VECTOR(0 to 1);
          led           :   OUT STD_LOGIC_VECTOR(0 to 15);
          seg           :   OUT STD_LOGIC_VECTOR(0 to 7);
          digit_enable  :   OUT STD_LOGIC_VECTOR(0 to 3));
end seven_segments;
```

Figure 6: Design's entity declaration



```
13  architecture Behavioral of seven_segments is
14      type SEGMENTS_DISPLAY is array(natural range 0 to 3) of STD_LOGIC_VECTOR(0 to 7);
15      type LED_DISPLAY is array(natural range 0 to 15) of STD_LOGIC_VECTOR(led'low to led'high);
16      type TEAMS_BIRTHDAYS is array(natural range<>) of SEGMENTS_DISPLAY;
17      type TEAMS_SEQUENCES is array(natural range <>) of LED_DISPLAY;
18
19      SIGNAL segment_CLK          :   STD_LOGIC := '0';
20      SIGNAL led_CLK              :   STD_LOGIC := '0';
21      SIGNAL seg_sequenceNumber   :   STD_LOGIC_VECTOR(1 downto 0) := "00";
22      SIGNAL led_sequenceNumber   :   STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

Figure 5: Architecture body's declaration section

Before its `BEGIN` keyword, the architecture body declared the 4, self-made array data types that were used to store everyone's birthday and sequence. This was shown in lines 14-17 of Figure 5.

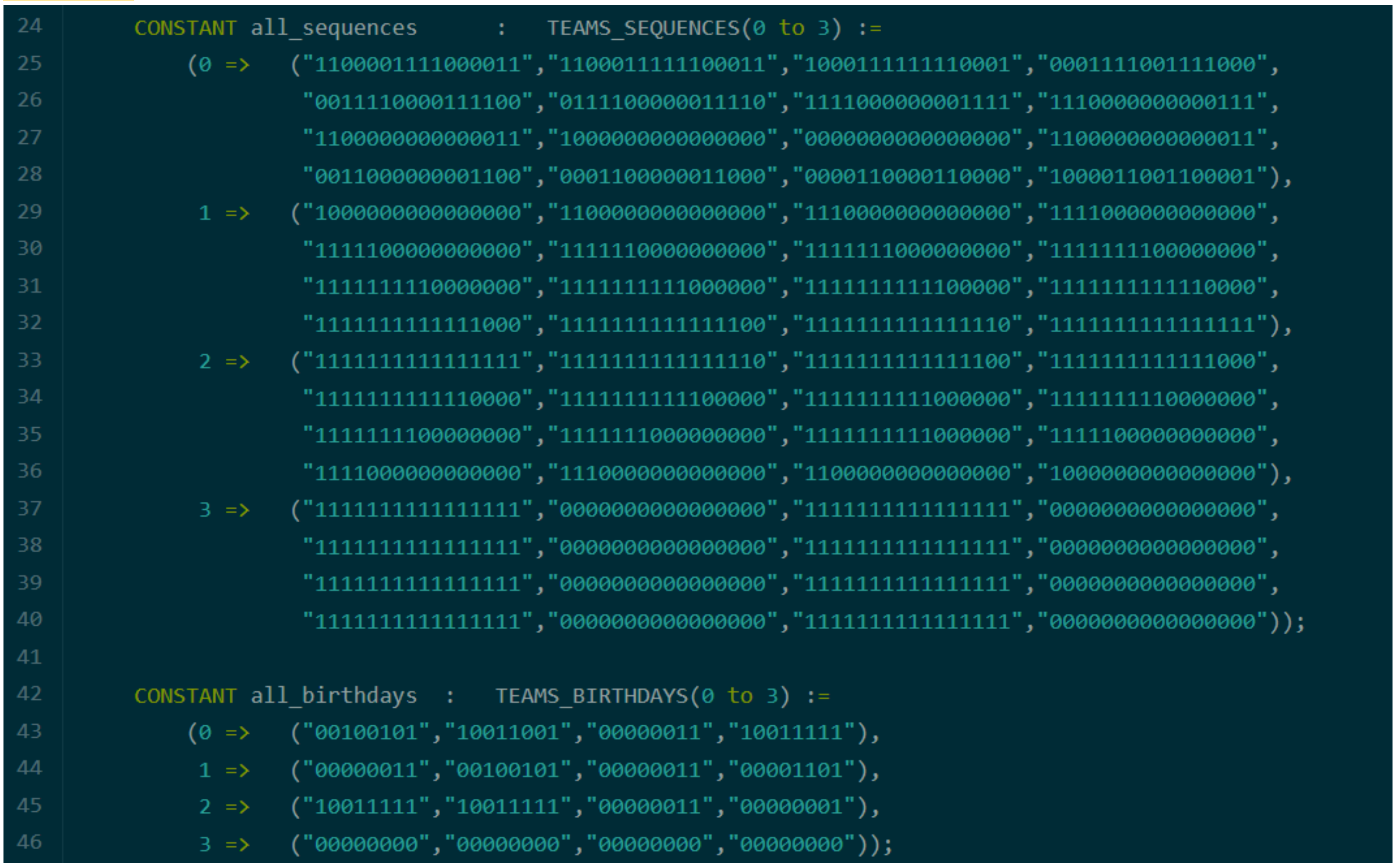
- Line 14 – Shows the `SEGMENTS_DISPLAY`. As seen in the line, each element was an 8-bit `STD_LOGIC_VECTOR` (each element was enough to hold the segment values for one digit). `SEGMENTS_DISPLAY` could hold 4 of these values. Therefore, one instance of it could store one 4-digit birthday.
- Line 15 – Shows the `LED_DISPLAY`. Each element was a `STD_LOGIC_VECTOR` where the lowest possible index was the lowest possible index of the led output port's `STD_LOGIC_VECTOR`. The highest possible indices were the same also. A literal index was not given to ensure that each bit of the `STD_LOGIC_VECTOR` would correspond to one bit of the led output port. Each of the `STD_LOGIC_VECTORS` encapsulated one pattern of a sequence. `LED_DISPLAY` held 16 of these and so it encapsulated the whole pattern.

- Line 16 – Created an array of `SEGMENTS_DISPLAY`. This allowed all birthdays of the team to be encapsulated in one instance of `TEAMS_BIRTHDAYS`
- Line 17 – Created an array of `LED_DISPLAY`, allowing all sequences of the team to be encapsulated in one instance of `TEAMS_SEQUENCES`

Following the data-type creations, different instances were declared and initialised:

- Lines 19 and 20 created the clock signals for the 7-segment displays and the LEDs respectively. Those were initialised to 0
- Line 21 created the signal used to track which digit of the 7-segment displays was on.
- Line 22 created the signal used to track which pattern of the sequence was to be displayed (as mentioned in "2.3.1.2 Tracking which Pattern to Display", p5). 4 bits were given to it to cover the 16 patterns in a sequence

Also, in the architecture's declaration, were constants that defined everyone's birthday and sequence. In Figure 7, instances of `TEAM_SEQUENCES` and `TEAMS_BIRTHDAYS` were created (with these instances being called `ALL_SEQUENCES` and `ALL_BIRTHDAYS` respectively).



```
    CONSTANT all_sequences      :   TEAMS_SEQUENCES(0 to 3) :=
        (0 =>   ("1100001111000011","1100011111100011","1000111111110001","0001111001111000",
                 "0011110000111100","0111100000011110","1111000000001111","1110000000000111",
                 "1100000000000011","1000000000000000","0000000000000000","1100000000000011",
                 "0011000000001100","0001100000011000","0000110000110000","1000011001100001"),
         1 =>   ("1000000000000000","1100000000000000","1110000000000000","1111000000000000",
                 "1111100000000000","1111110000000000","1111111000000000","1111111100000000",
                 "1111111110000000","1111111111000000","1111111111100000","1111111111110000",
                 "1111111111111000","1111111111111100","1111111111111110","1111111111111111"),
         2 =>   ("1111111111111111","1111111111111110","1111111111111100","1111111111111000",
                 "1111111111110000","1111111111100000","1111111111000000","1111111110000000",
                 "1111111100000000","1111111000000000","1111111111000000","1111100000000000",
                 "1111000000000000","1110000000000000","1100000000000000","1000000000000000"),
         3 =>   ("1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000"));

    CONSTANT all_birthdays  :   TEAMS_BIRTHDAYS(0 to 3) :=
        (0 =>   ("00100101","10011001","00000011","10011111"),
         1 =>   ("00000011","00100101","00000011","00001101"),
         2 =>   ("10011111","10011111","00000011","00000001"),
         3 =>   ("00000000","00000000","00000000","00000000"));
```

Figure 7: Constants used to define the team's birthdays and sequences

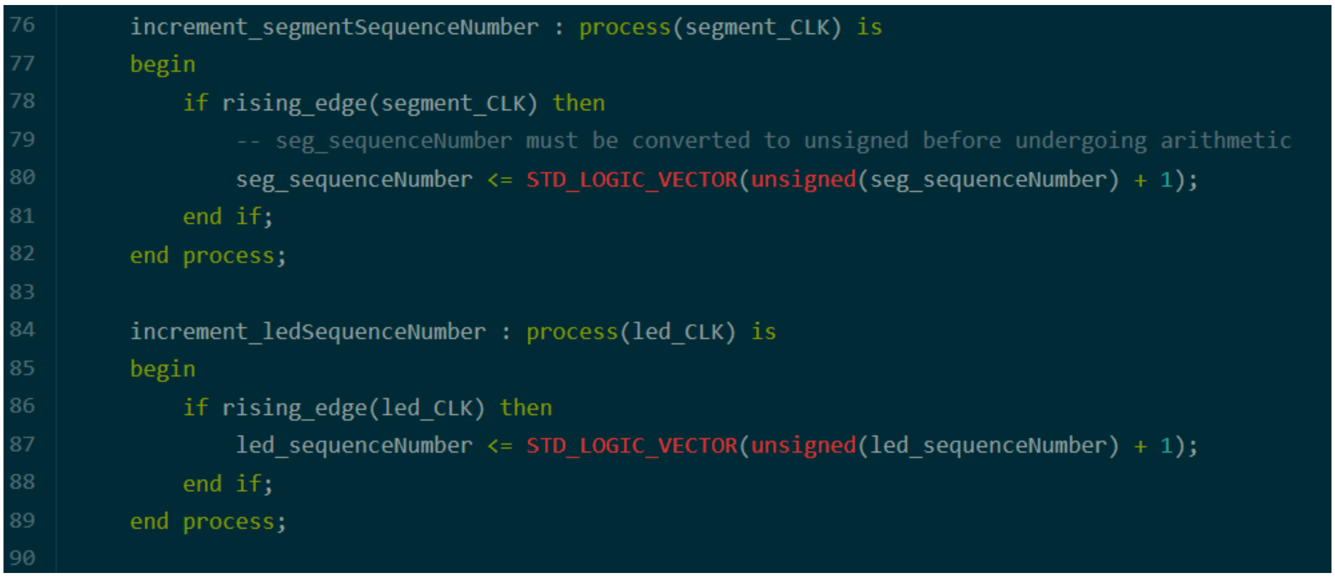
It was important to realise that to activate a particular segment, it had to receive a LOW signal as Section 2.3.2.1 (p5) showed that these `STD_LOGIC_VECTORS` stored the value of the segments' cathodes. The convention of Figure 2 (p3) was used to determine which of those bits corresponded to which segment. Line 14 of Figure 5, has the `STD_LOGIC_VECTORS` with the index range of `(0 TO 7)`. Therefore, the leftmost bit of the `STD_LOGIC_VECTOR` had an index of 0. These facts were reflected in the initialisation.

The two instances were made constants to prevent unintentional modifications. Furthermore, the design requirements did not require this data to be changed.

3.1 The Architecture Body

The first two processes in the architecture were the clock dividers for the LED sequence and 7-segment displays. These processes were discussed in Sections 2.3.1.1 - Clock Divider and 2.3.2.1 - The Idiosyncrasies of the Basys-3's 7-Segment Displays.

The next two processes were in Figure 8. The process `INCREMENT_LEDSEQUENCENUMBER` had a sensitivity list with only `LED_CLK`. Hence, it would only execute when there was an event on `LED_CLK`. Line 78, shows that the increment of line 80 only runs if the event was a rising edge. However, `SEG_SEQUENCENUMBER` was a `STD_LOGIC_VECTOR` and so arithmetic could not be performed on it. In order to increment by 1, it first had to be converted into an unsigned data type. Once this converted value was incremented, it was reconverted to a `STD_LOGIC_VECTOR`.

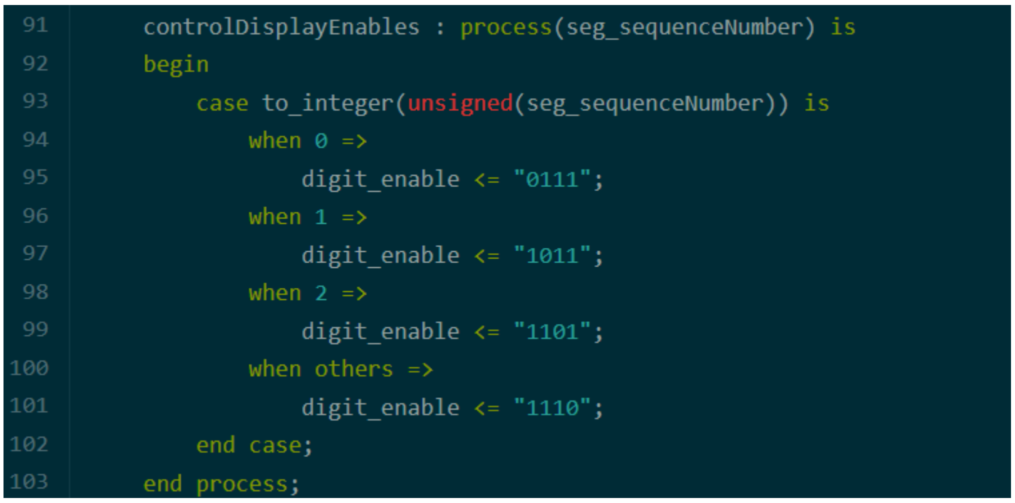


```
76        increment_segmentSequenceNumber : process(segment_CLK) is
77        begin
78            if rising_edge(segment_CLK) then
79                -- seg_sequenceNumber must be converted to unsigned before undergoing arithmetic
80                seg_sequenceNumber <= STD_LOGIC_VECTOR(unsigned(seg_sequenceNumber) + 1);
81            end if;
82        end process;
83
84        increment_ledSequenceNumber : process(led_CLK) is
85        begin
86            if rising_edge(led_CLK) then
87                led_sequenceNumber <= STD_LOGIC_VECTOR(unsigned(led_sequenceNumber) + 1);
88            end if;
89        end process;
90
```

Figure 8: Processes that incremented the sequenceNumber signals

A neat feature of this line happened when `SEG_SEQUENCENUMBER` incremented from $11_{\text{bin}} = 3_{\text{dec}}$ to become $100_{\text{bin}} = 4_{\text{dec}}$. However, `SEG_SEQUENCENUMBER` was a 2-bit number. When this value was incremented, only the 2 least-significant bits of the result were used, which effectively overflowed the result and reset the signal to 0. After, the third element of the `SEGMENTS_DISPLAY` was accessed, the zeroth element would be accessed. This restarted the cycle through the 7-segment digits, as was desired. The process of `INCREMENT_LEDSEQUENCENUMBER` worked in the same way.

The next process of `controlDisplayEnables` (of Figure 9) had the responsibility of enabling and disabling the 7-segment displays, depending on which digit was being shown. It did so by first looking at the value of the index `seg_sequenceNumber`.



```vhdl
    controlDisplayEnables : process(seg_sequenceNumber) is
    begin
        case to_integer(unsigned(seg_sequenceNumber)) is
            when 0 =>
                digit_enable <= "0111";
            when 1 =>
                digit_enable <= "1011";
            when 2 =>
                digit_enable <= "1101";
            when others =>
                digit_enable <= "1110";
        end case;
    end process;
```

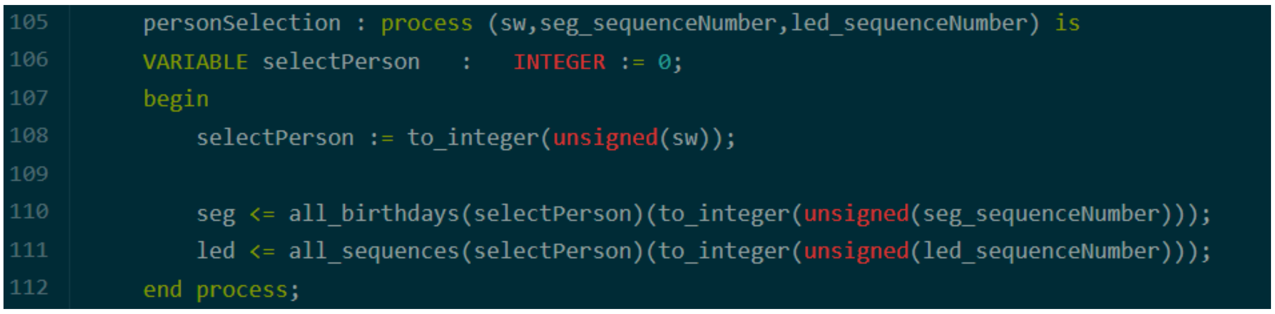
Figure 9: The controlDisplayEnables process

The `digit_enable` port was controlled the anodes of each of the displays. In spite of this, a LOW signal was required to activate a particular display since the basys-3 board did not allow for direct access to those anodes. Rather, access was made available to a transistor which blocked or allowed voltage into the anode. For that reason, each of the binary literals assigned to `digit_enable` had only 1 LOW bit.

The final process was that of Figure 10, which was what made the assignments to ports. Line 106 created the local variable `selectPerson` was used an index into the largest arrays of `all_sequences` and `all_birthdays`. Line 108 showed that this variable's value was a direct conversion of the 2-bit input `sw` into an integer[1]. An integer was required as only those data types can be used as an index.

In lines 110 and 111, each of these indexes returned another array (`segments_display` and `led_display`, the four digits for the birthday and the 16 patterns of the sequence respectively). These arrays could not be assigned to the output ports; only their elements could. The `sequenceNumbers` were used as these indices once they were converted to integers.

The process has a sensitivity list with the inputs and these indices, as the outputs should only change if any of those change.



```vhdl
105      personSelection : process (sw,seg_sequenceNumber,led_sequenceNumber) is
106      VARIABLE selectPerson    :    INTEGER := 0;
107      begin
108          selectPerson := to_integer(unsigned(sw));
109
110          seg <= all_birthdays(selectPerson)(to_integer(unsigned(seg_sequenceNumber)));
111          led <= all_sequences(selectPerson)(to_integer(unsigned(led_sequenceNumber)));
112      end process;
```

Figure 10: The personSelection process

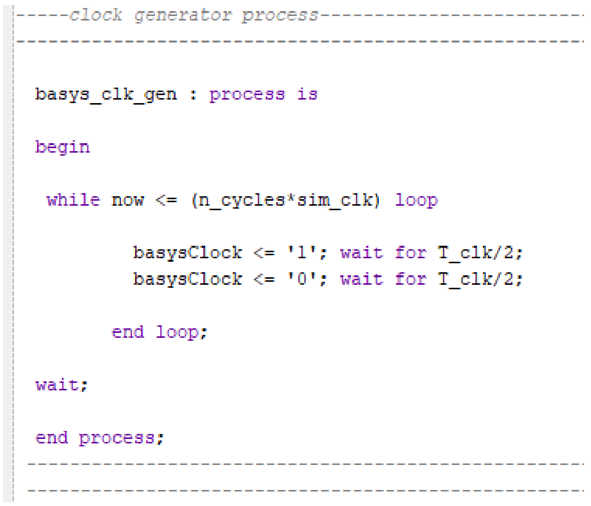
[1] A conversion to `unsigned` was first needed as there was no direct conversion to integer from a `std_logic_vector` type

4 Simulation and Hardware Testing

4.1 Simulation

4.1.1 Testing Strategy

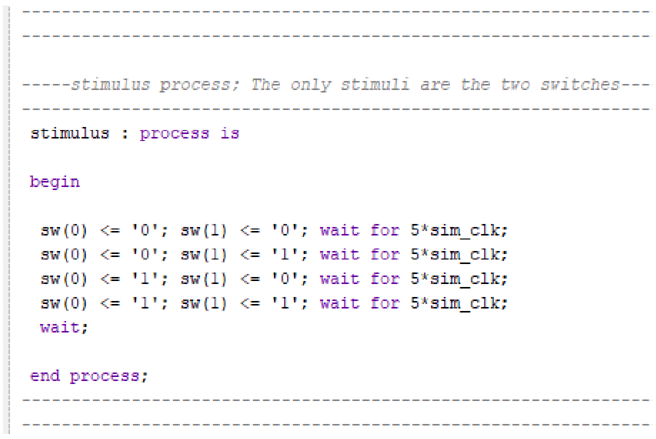
The testing strategy for the design was concise and easily relatable to the functionality. The main design only has two inputs therefore the test bench only consisted of two processes; one to control each. The first part of the testing strategy consisted of a simple counter that alternated the main Basys clock input between 0 and 1. The process responsible is shown in Figure 11.



```
-----clock generator process------------------------
----------------------------------------------------

 basys_clk_gen : process is

 begin

  while now <= (n_cycles*sim_clk) loop

          basysClock <= '1'; wait for T_clk/2;
          basysClock <= '0'; wait for T_clk/2;

        end loop;

 wait;

 end process;
 ---------------------------------------------------
 ---------------------------------------------------
```

Figure 11: The clock generator process

The second part involved setting the two switch variables to all four of their possible combinations. Sufficient waiting time was left in between the different combinations to allow the design enough time to produce results for each birthday index. The switch process is shown in Figure 12.



```
----------------------------------------------------------
----------------------------------------------------------

-----stimulus process; The only stimuli are the two switches---
----------------------------------------------------------
 stimulus : process is

 begin

  sw(0) <= '0'; sw(1) <= '0'; wait for 5*sim_clk;
  sw(0) <= '0'; sw(1) <= '1'; wait for 5*sim_clk;
  sw(0) <= '1'; sw(1) <= '0'; wait for 5*sim_clk;
  sw(0) <= '1'; sw(1) <= '1'; wait for 5*sim_clk;
  wait;

 end process;
----------------------------------------------------------
----------------------------------------------------------
```

Figure 12: The switch combination stimuli

With this testing strategy the simulation would highlight the changing values of:

- The LED sequences
- The 7-Segment display sequences
- The 7-Segment control section

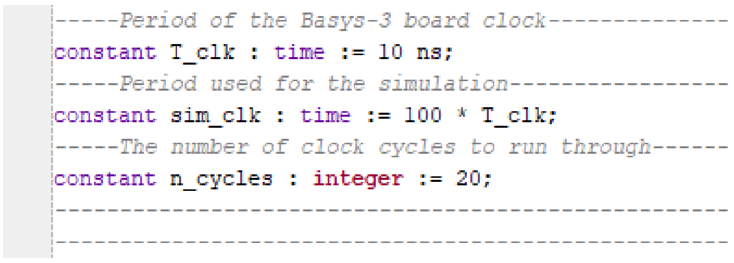
For each of the four possible values of the switch inputs.

4.1.2 Testing Settings

To ensure that the simulation was meaningful and easily readable certain values were chosen for the testing settings. Within the testbench itself three values were specified:

- The period of the real Basys-3 clock
- An altered period for the simulation
- A set number of cycles for the simulation

The constants within the testbench are shown in Figure 13.



```
-----Period of the Basys-3 board clock--------------
constant T_clk : time := 10 ns;
-----Period used for the simulation-----------------
constant sim_clk : time := 100 * T_clk;
-----The number of clock cycles to run through------
constant n_cycles : integer := 20;
----------------------------------------------------
----------------------------------------------------
```

Figure 13: Testbench constants

As shown within the while loop condition in Figure 12, the clock process runs for the length of time determined by the product of the number of cycles `N_CYCLES` and the simulation clock period `SIM_CLK`. The simulation was designed to run for a length of 20µs as shown by Equation (8).

$$\text{Simulation Length} = \text{sim}_{\text{clk}} \times \text{n}_{\text{cycles}} = 100 \times 10 \times 10^{-9} \times 20 = 20\mu\text{s} \qquad (8)$$

The simulation has four switch combinations to test for. Figure 12 details that the waiting time in between each combination is 5 times the simulation period. This value was decided as the simulation period is 1µs therefore if each switch combination runs for 5µs this will fit exactly into the 20µs simulation time as shown in Equation (9).

$$5 \times \text{sim}_{\text{clk}} \times \text{switch combinations} = 5 \times (100 \times 10^{-9}) \times 4 = 20\mu\text{s} \qquad (9)$$

Within the simulation settings the simulation length was set to 20µs to match the value calculated for the testbench.

It was important that two additional values were added to the design for simulation purposes. As the values of `MAXCOUNT_LED` and `MAXCOUNT` were still set to high values for real world human interaction; they were changed to 8 and 32 respectively so that they could operate within the simulations micro second time frame.

4.1.3 Results and Discussion

First it was checked that the stimuli worked as intended. Figure 14 shows how the switch values change over the 20µs.

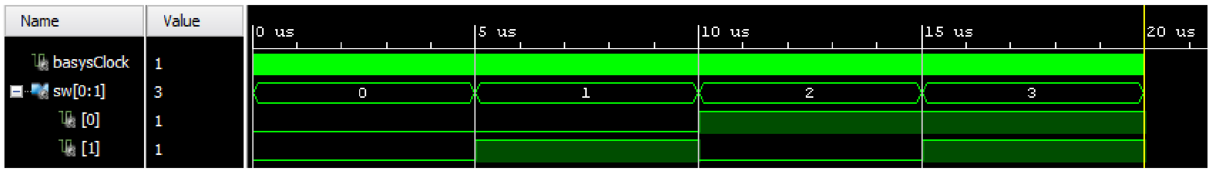


Figure 14: Switch values over the 20µs

The simulation showed the four different switch values changing over time as hoped for. This part of the simulation was essential as all the other sections depend on the value of the switches to provide different results.

Next the LED sequences were observed. It was expected that the LEDs would move through the values set in their constants with different patterns being created for each switch setting. The results are shown in Figure 15.

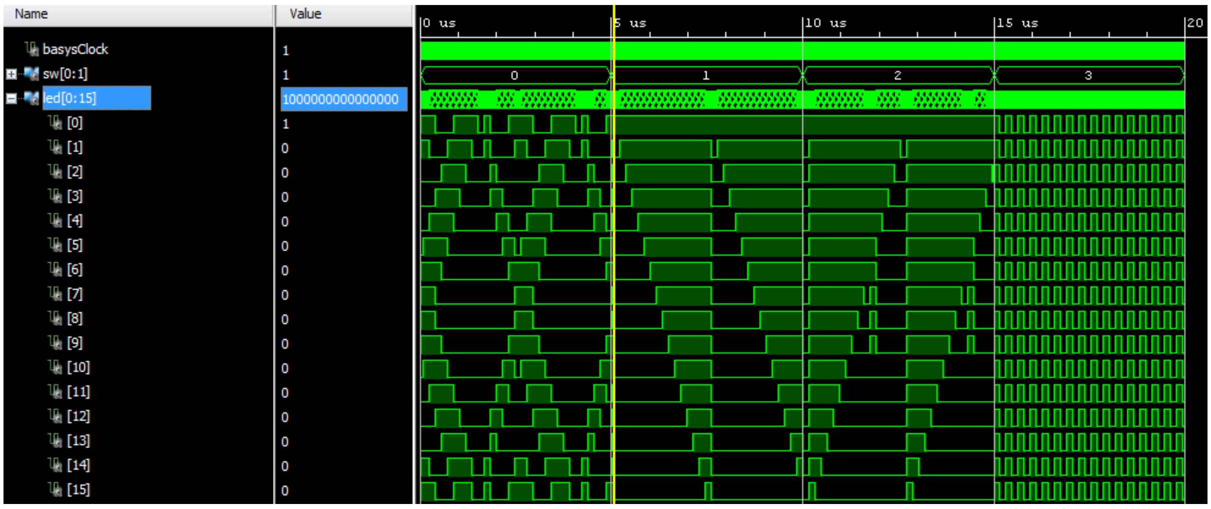


Figure 15: LED values over the 20µs

The simulation shows that the LED values were traversed on the clock as expected. There is a different pattern shown for each of the switch values which shows that the indexing of the LED sequences has linked with the changing binary values of the switch combinations. From inspecting the first LED sequence value for the switch of value “01” it can be clearly seen in the panel on the left of Figure 15 that it has been set to “1000000000000000”. This value can then be compared with the first sequence value in the constant declaration in Figure 7; both values are the same indicating that the simulation chooses the correct set of LED sequences for the given switch. The patterns are all repeated twice within the 5µs time frame of their switch proving that the sequence loops back around and does not simply stop and remain on the last value.

The segment display was tested in a similar fashion. The results are shown in Figure 16.

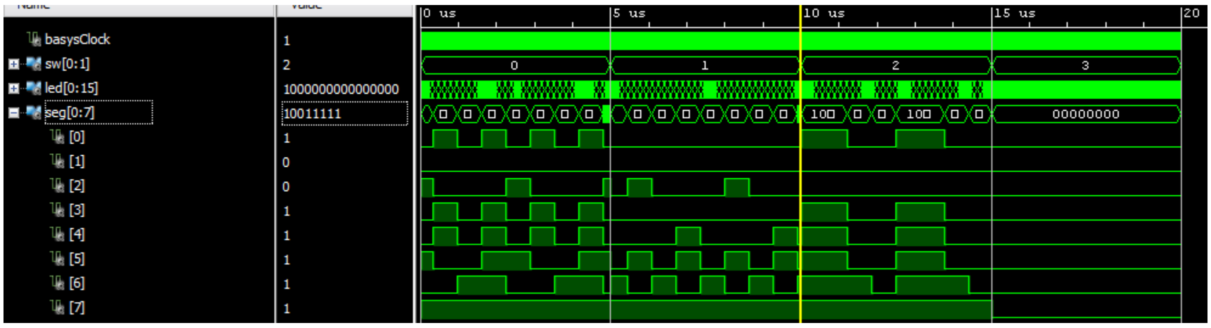
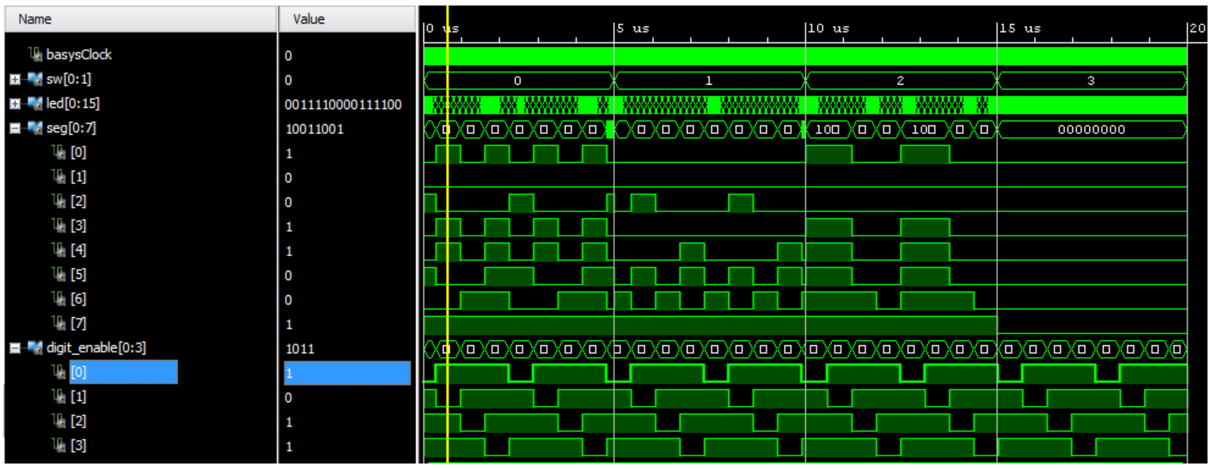


Figure 16: Segment display values over the 20µs

Figure 16 shows that there are four distinct segment combinations for each of the four distinct switch values; this is in keeping with the design specification. This shows that the segment part of the design has correctly communicated with the switch value section. The left-hand side panel can be checked to determine if the correct list of segment values has been chosen for the specific switch. The switch of index 2 has been selected in Figure 15 with the vertical yellow line placed on its first 7 segment value. The segment value for the switch reads as “10011111”, consultation with Figure 7 reveals that this is the correct value for the switch of index 2 proving that the switches select the correct segment data set.

The resulting simulation for the data segment control section is shown in Figure 17.



Name	Value
basysClock	0
sw[0:1]	0
led[0:15]	0011110000111100
seg[0:7]	10011001
[0]	1
[1]	0
[2]	0
[3]	1
[4]	1
[5]	0
[6]	0
[7]	1
digit_enable[0:3]	1011
[0]	1
[1]	0
[2]	1
[3]	1

Figure 17: Segment control section over the 20µs

Figure 17 demonstrates that the value to choose which segment display to operate changes over time as expected. To check that the correct segment is on for the correct 7-segment digit the values of 7-segment section and 7-segment control section can be compared. The vertical yellow line in Figure 17 indicates a point where the 7-segment control value is “1011”; this means that the second segment display would be active and therefore the second value in the chosen switches data set is needed. At the same point in the simulation the segment value itself is given as “10011001” in the left-hand-side panel. Figure 7 shows that – for the switch of index 0 – the second segment value in the data set is also “10011001”. This proves that the correct segment value is matched to the correct segment control value. The segment control values follow the same pattern no matter which switch is involved; this is accurate as the order of which segment display is high is the same no matter which switch is selected or what digit it is going to be displaying.

Overall the whole simulation was as expected. For each given switch value there was a different LED pattern and different segment display values while the segment selector section maintained that the correct digit is on when the corresponding segment display is in use.

4.2 Hardware

4.2.1 Board diagram

As the design was simulated successfully it was then tested on the Basys board itself. Three different class of components were used:

- Four 7-Segment Displays
- Sixteen LEDs
- Two Switches

The diagram shown in Figure 18 highlights which components were used on the hardware.

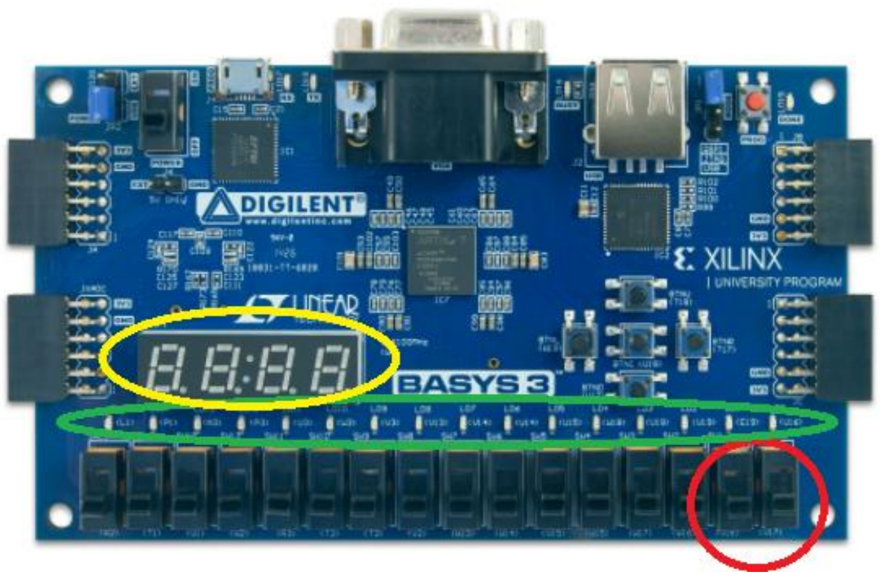


Figure 18: Components used highlighted on the Basys board. Red:Switches, Green:LEDs, Yellow:7-SegmentDisplay [2]

4.2.2 Testing and Results

The testing strategy for the hardware was much like the testing strategy for the simulation. The clock process is internal so did not need to be considered therefore the only variable was the two switches shown in red in Figure 18. The two switches were set to represent “00”,”01”,”10” and “11” much like the testbench with the output on the LED’s (shown in green) and the four 7-segment displays (shown in yellow) was observed. Table 2 summarises the possible inputs to the board and the expected outputs.

Table 2: Summary of potential inputs and expected outputs

Team Member	**Switch Value**	**Segment Output**
Jee Ken Chung	00	2401
Fraser Rigby	01	0203
Steven Webster	10	1108
ERROR DISPLAY	11	8.8.8.8.

When the board was first programmed the switches were both low and consequently representing “00”. The switches were then set to the three other combinations with the board being examined in between. The outputs on the board for each of the four input combinations are shown in Figures 19, 20, 21 and 22.

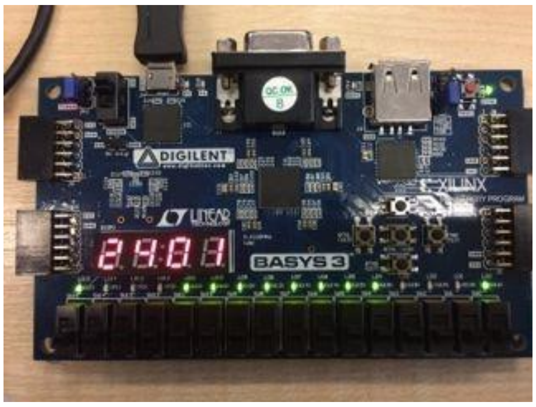


Figure 19: Board output for "00"

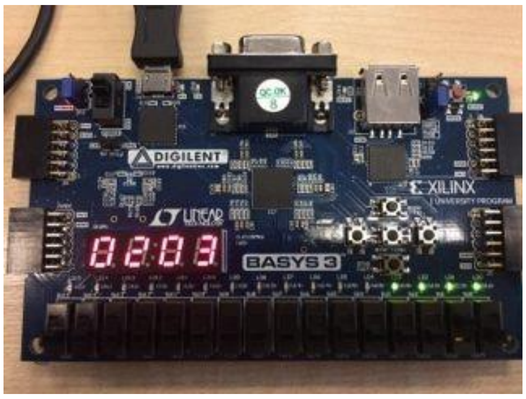


Figure 20: Board output for "01"

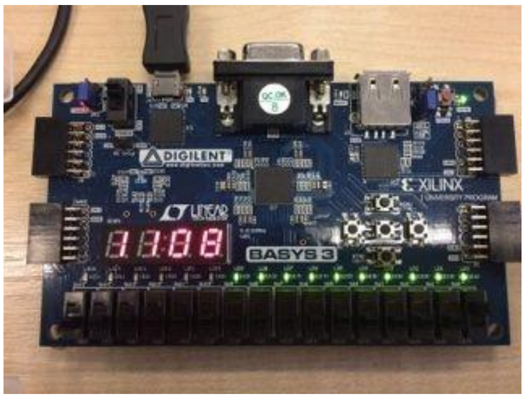


Figure 21: Board output for "10"

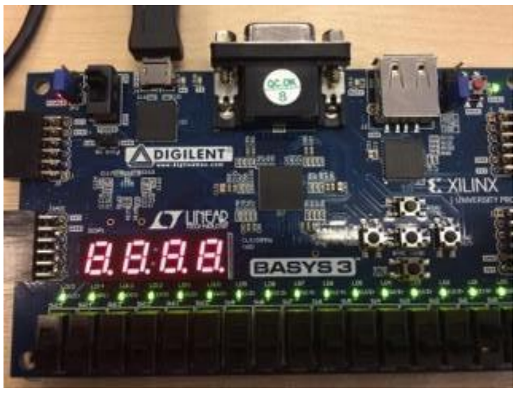


Figure 22: Board output for "11"

As can be seen by inspecting Figure 19 to Figure 22 and comparing them to the output digits stated in Table 2, the board produced the expected results. For each of the four values of the switches the 7-segment displays exhibited the correct digit. The LED patterns were also displayed for the correct corresponding switch.

5 Synthesis and Implementation Results

After synthesis and implementation, a floor plan of the design was produced. The floor plan is shown in Figure 23 while a zoomed in version is shown in Figure 24.

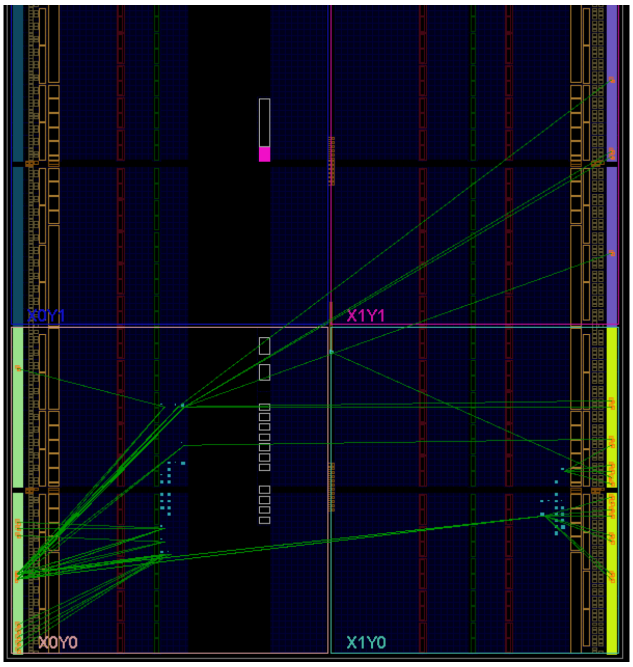


Figure 23: Floor plan of the design

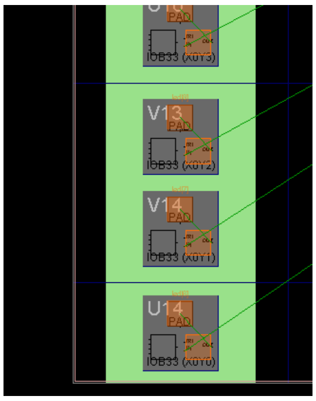
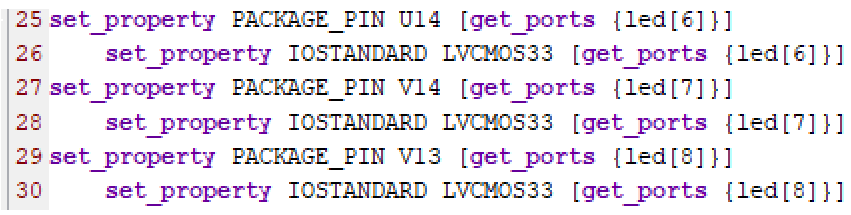


Figure 24: Expanded floor plan

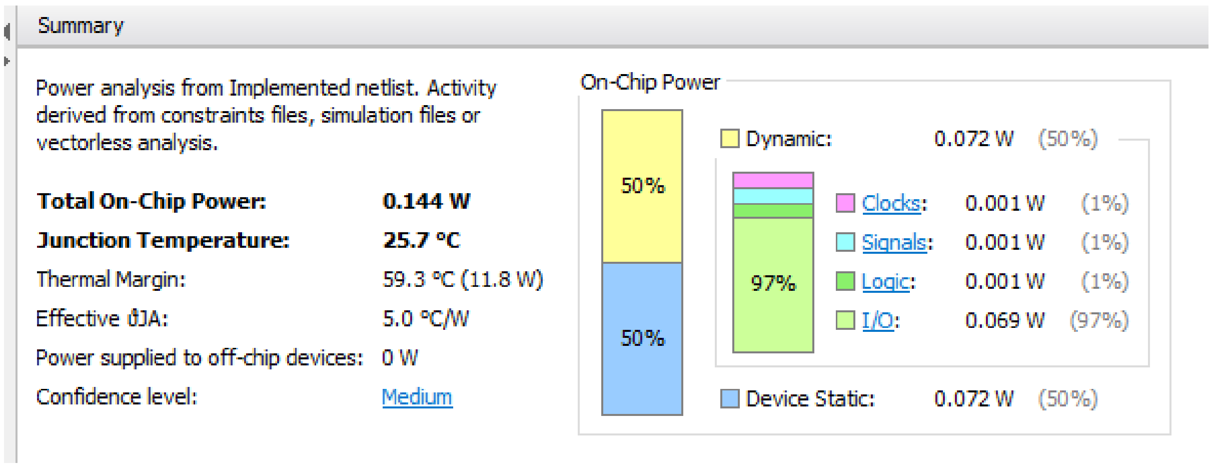
The pin names shown to be in use in Figures 23 and 24 match all the pin names used in the constraints file for the design; this indicates that the synthesis and implementation were completed successfully. A portion of the constraints file is shown in Figure 25, it features the same pins seen in Figure 24. All these pins are for LEDs and are therefore shown to be linked to some of the different LED values from the design.



```
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
```

Figure 25: Portion of the constraints file showing pins from the floor plan

The software allows for detailed analysis of the power usage of any synthesised and implemented design. A summary of power usage is shown in Figure 26.


Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

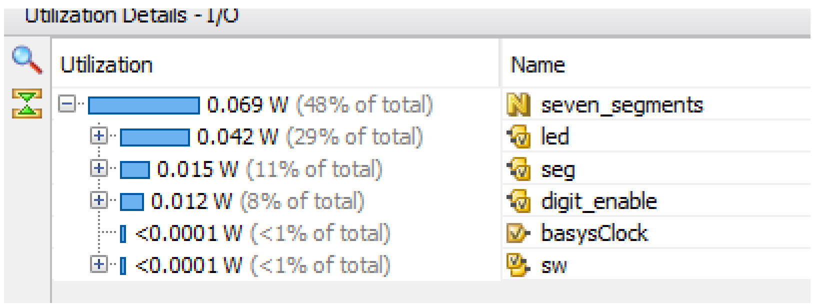
Total On-Chip Power:	**0.144 W**
Junction Temperature:	**25.7 °C**
Thermal Margin:	59.3 °C (11.8 W)
Effective ϑJA:	5.0 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

On-Chip Power

Dynamic:	0.072 W	(50%)
Clocks:	0.001 W	(1%)
Signals:	0.001 W	(1%)
Logic:	0.001 W	(1%)
I/O:	0.069 W	(97%)
Device Static:	0.072 W	(50%)

Figure 26: Power Summary of the design implementation

This summary details that most of the power supplied to the FGPA is used by the input and outputs. The “input and output” section consumes 97% of the power supplied while the Clocks, Signals and Logic sections only consume 1% each. This reveals how little power the details of the created design use as all the logic coded into the design is such a small percentage. The summary shows how the effort to physically light the LEDs and 7-segment display is far more exhaustive than any of the calculations or the forms of control coded within. The power summary can be further explored to reveal the specific power consumption of each component type in the I/O section. This further detail is shown in Figure 27.


Utilization Details - I/O

Utilization	Name
0.069 W (48% of total)	seven_segments
0.042 W (29% of total)	led
0.015 W (11% of total)	seg
0.012 W (8% of total)	digit_enable
<0.0001 W (<1% of total)	basysClock
<0.0001 W (<1% of total)	sw

Figure 27: Power break down for the I/O components

This further breakdown of the input and output components reveals that the LEDs are consuming the most power with 29% of the total power consumption attributed to them. This is to be expected as there are sixteen LEDs in use in the design while only four 7-segment displays. It is also unsurprising that the 7-segment display uses the next most power with 11% of the total as they also have a lighting component like the LEDs.

6 Organisation of the project and team working aspect

This project was to be completed in groups. This was advantageous as it allowed for the different tasks that were required to be completed to be divided up between the members of the group and also allowed for each member of the group to in essence, play to their strengths in terms of which aspects of the project they felt most comfortable about attempting.

It was also crucial that the project was effectively planned. A series of group meetings were planned in order to discuss the project where it was agreed to try and finish writing the code and then implement in on the Basys-3 board well in advance of the code demonstration in order to try and alter in slight errors that may have occurred. This also meant that report writing could be started in good time and would not have to be rushed in order to comply with the deadline.

In terms of the VHDL code, this was done by dividing the tasks into the actual implementation of the VHDL code, obtaining simulation results and then implementing the design on to the FPGA board as this was the logical order in which to undertake the project.

As planned, the code was written and programmed on to the FPGA board in very good time as well as obtaining simulation results. This allowed the group to meet up and discuss the various processes that had been completed so that everyone had a good enough understanding to construct the various sections of the report and also to take part in the demonstration of how the code worked.

The report was divided in much the same way, with each member if the group assigned to discuss their section of the report, with the additional sections that did not concern the design process being divided up equally. It was also decided that it would be best to complete the report over One Drive in order to ensure that each member of the group could alter the report at their own will.

It also proved very advantageous to have the design process completed in good time as this allowed for more time to write the report and also allowed more time for any additional research to be carried out.

During this task, it was important to have good communication between the group members to ensure that the work could be completed in the most efficient manner. The group did communicate well as it was clear at all times what tasks had to be completed and who was in charge of completing them. This optimised the time spent working on writing the code and implementing it onto the FPGA board and ensured that time wasn't wasted debating the next course of action or trying to split up the tasks (as previously mentioned it was made clear how the different processes that were required to be completed would be split up amongst the group).

Overall, the group worked well together, as the tasks were successfully divided amongst the members which gave rise to an efficient working process which meant that the overall project was completed in very good time which enabled the group members to educate each other on the separate components of the project. This was particularly effective as it allowed for a successful demonstration of the code to be undertaken which showed each member's birthday being displayed on the Basys-3 board whilst an individual LED sequence executed simultaneously.

7 Conclusion

This group project involved practical VHDL design and implementation on a Basys-3 FPGA board. The task was to design a VHDL system that would perform two functions on the board. These functions were:

- To use the 7 segment display in order to illuminate the birthdays of the group members. The birthdays were to be illuminated in the form DDMM where D is the day and M is the month, so for example if one of the birthdays was the 15th of June then the 7 segment display would show 1506. The order in which the birthdays was displayed was to be determined using either switches or buttons on the board, in this case switches were used to avoid bouncing.
- In addition and simultaneously to the birthdays being displayed, the LEDs on the board would be illuminated thanks to individually designed sequences. This meant that the LED sequence executed when the 1st group member's birthday was being displayed was to be different to the LED sequence displayed when 2nd group member's birthday was being displayed and so on.

VHDL code was written in order to achieve these outcomes and the full code is attached as an appendix. As mentioned in the top down discussion of the VHDL code, each member of the group was assigned a 2-bit number and this allowed just two switches on the board to be put into use and each switch acted as a single bit of the 2-biut number. Three 2-bit numbers would represent each of the group members and a final 2-bit number would represent an error.

Due to the switches being considered as bits to a binary number this allowed for them to be expressed as a 2-bit `STD_LOGIC_VECTOR`.

The entity declarations and architecture body declarations are displayed in section 3.2 and are depicted in Figures 5 and 6. A description is provided as to the role that each line of code plays in the overall design. In addition to this, the constants that were employed to define each member's birthday and LED sequence are displayed, in Figure 7. It was also determined that in order to activate any particular segment on the 7 segment display, a LOW signal had to be received.

Section 3.3 discusses the architecture body in detail. It was mentioned that processes were required for clock dividers for both the LED sequence and the 7-segment display, which were discussed previously in the report. The different stages of the architecture body are provided in Figures 8, 9 and 10, again with a description of the importance of each process in the total design.

In order to test the code testing strategies had to be developed. The testing strategies were similar for both the software and the hardware.

As the main design had just two inputs this mean that it was only required to have one process controlling each input, meaning that the test bench consisted of two processes. This mean that it was fairly succinct. A counter was created for the basys clock input and the process for this is displayed in Figure 11.

In addition to this, the switch variables had to be set to all possible combinations (4 in this case) and when this was done it was crucial that there was enough waiting time in between the different combinations in order for results to be produced for each birthday index.

The test settings were decided upon in order to make sure that the generated results could be easily read and analysed. Three constants were defined which are displayed in Figure 13. These constants were the period of the basys clock, the period of the simulation and the number of clock cycles that the simulation was run for. These constants were used along with equation (8) to determine the simulation run time.

Two additional values were then added to the design before simulation with reasons for this being provided in section 4.1.2.

A simulation was run with the results being displayed in section 4.1.3. It is clear from inspection of the results that the simulation was successful. Figure 14 showed the 4 different switch values changing over the required time period.

Figure 15 then shows the LED sequences. It is clear that, as expected the LEDs were traversed on the clock. A different pattern is displayed and this illustrates that the execution of the LED sequences matched with the changing switch combinations. In addition, patterns of the LEDs all repeated themselves twice which was desirable as it conveyed that the sequences loops back around and does not simply stop executing and remain on the previous value.

From examining Figure 16 it can be seen four segment combinations were achieved for each of the four switch values. This again indicated that the simulation had been successful as it was evident that the segment portion of the design matched up with the switch value portion of the design.

From Figure 17 it can be verified that the value to choose which segment display operates varies over time. This was the expected result which again cemented the fact that the simulation was a success.

Overall, the entire simulation can be regarded as successful, as, each result represented in the figures were exactly what they were expected to be when the design was created.

The next stage was to test the design on the board. For this, the only variable that had to be taken into account was the two switches that were being utilised on the board. The two switches which were used are shown in Figure 18. As previously stated, the switches represented a 2-bit number. The inputs and expected outputs are displayed in Table 2. When the testing was done and both of the switches were kept low representing the input “00”, the 7-segment display showed 2401. The inputs were then changed to represent the other combinations shown in Table 2 and the resulting display was always as expected. Therefore, the hardware testing was successful.

Overall, this project can be considered as a success. This is evident from inspecting both the software and hardware results. All of the simulation results completely agreed with what was expected. Then, when the testing was done on the basys-3 board, for each input combination, the 7-segment display showed exactly what was desired as well as the required LED sequence being executed. Therefore, this was a successful project.

Some potential further work that could be to use the buttons available on the board rather than the switches. This could lead to further work in terms of investigating ‘bouncing’ effects which were previously mentioned.

8 References

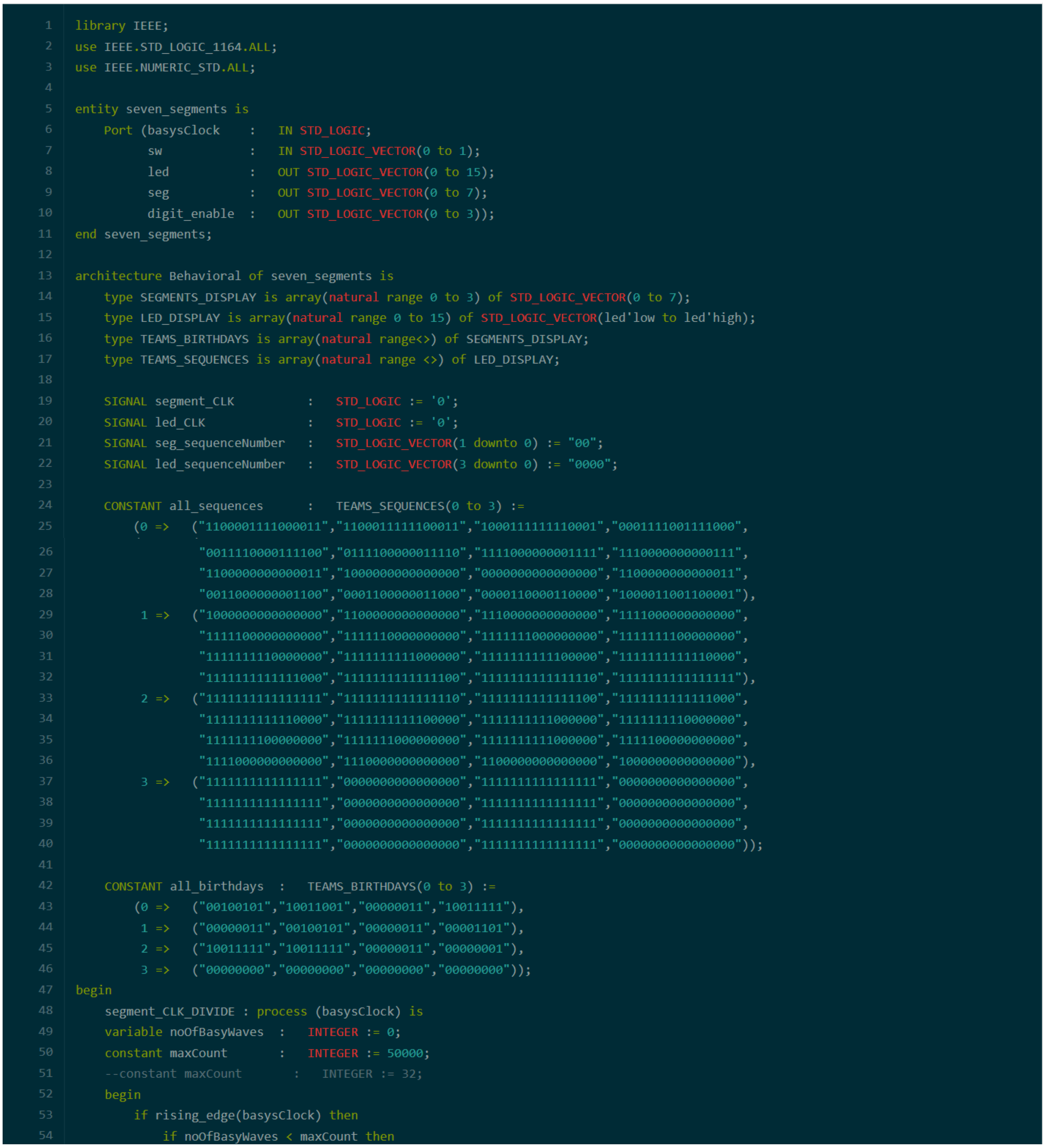
[1] L. Crockett, “RE: EE270 - Multiple Inputs Being High,” University of Strathclyde, Glasgow, 2018.

[2] Digilent, “Basys 3 FPGA Board Reference Manual,” Pullman, 2016.

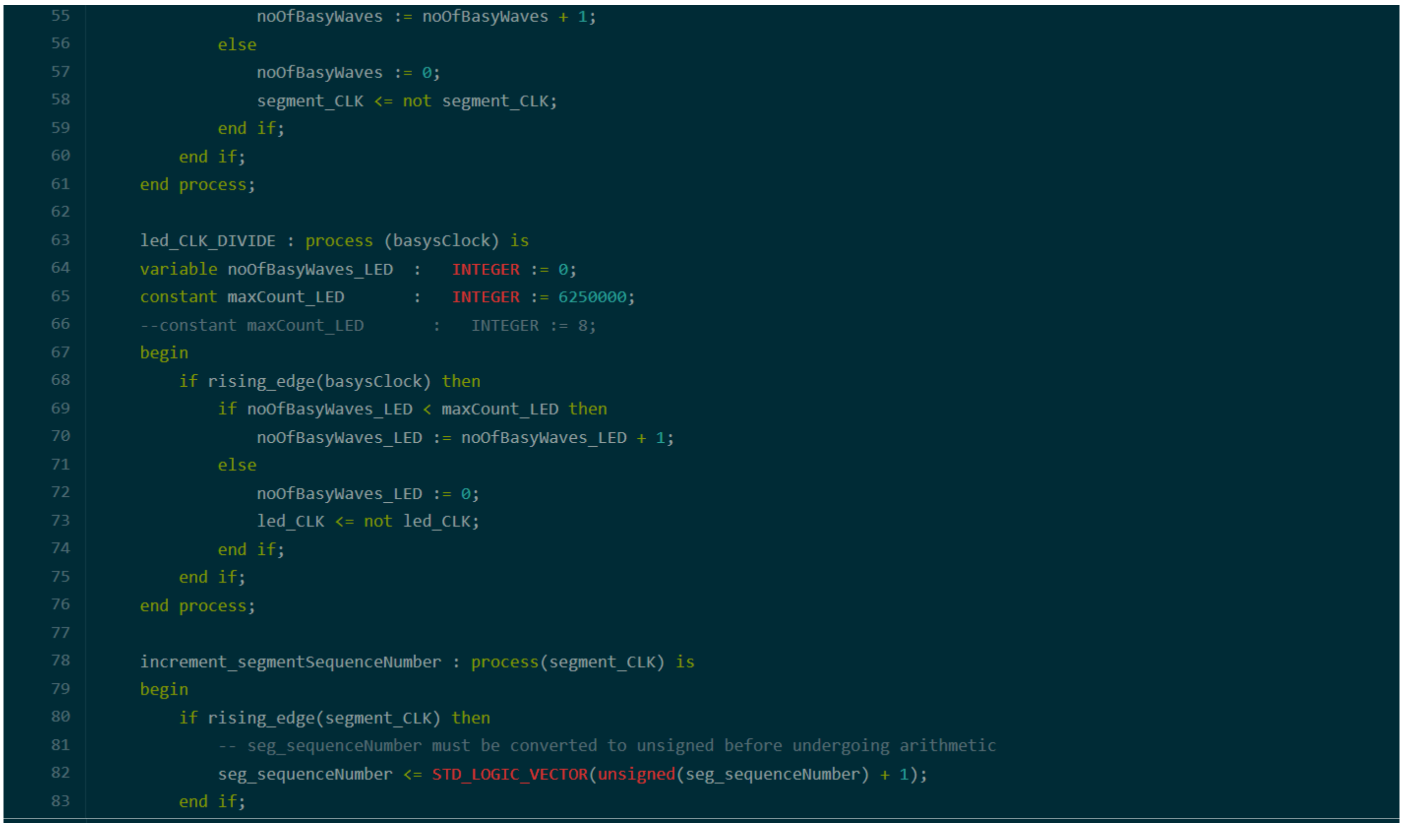
[3] J. K. Chung, “EE270 Semester 2: Laboratory Logbook,” Glasgow, 2018.

9 Appendices

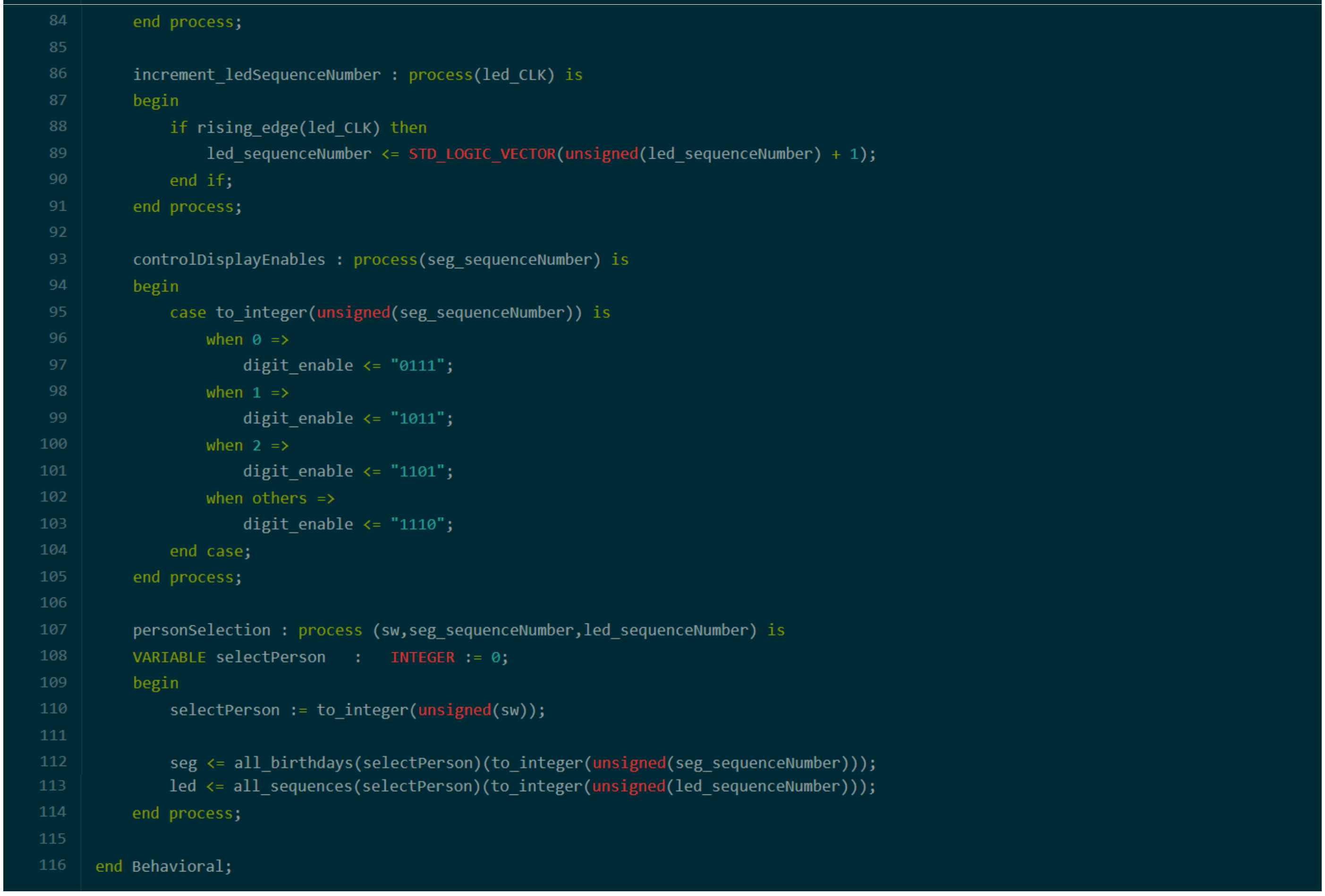
9.1 Design Source Code



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

entity seven_segments is
    Port (basysClock    :   IN STD_LOGIC;
          sw            :   IN STD_LOGIC_VECTOR(0 to 1);
          led           :   OUT STD_LOGIC_VECTOR(0 to 15);
          seg           :   OUT STD_LOGIC_VECTOR(0 to 7);
          digit_enable  :   OUT STD_LOGIC_VECTOR(0 to 3));
end seven_segments;

architecture Behavioral of seven_segments is
    type SEGMENTS_DISPLAY is array(natural range 0 to 3) of STD_LOGIC_VECTOR(0 to 7);
    type LED_DISPLAY is array(natural range 0 to 15) of STD_LOGIC_VECTOR(led'low to led'high);
    type TEAMS_BIRTHDAYS is array(natural range<>) of SEGMENTS_DISPLAY;
    type TEAMS_SEQUENCES is array(natural range <>) of LED_DISPLAY;

    SIGNAL segment_CLK          :   STD_LOGIC := '0';
    SIGNAL led_CLK              :   STD_LOGIC := '0';
    SIGNAL seg_sequenceNumber   :   STD_LOGIC_VECTOR(1 downto 0) := "00";
    SIGNAL led_sequenceNumber   :   STD_LOGIC_VECTOR(3 downto 0) := "0000";

    CONSTANT all_sequences      :   TEAMS_SEQUENCES(0 to 3) :=
        (0 =>   ("1100001111000011","1100011111100011","1000111111110001","0001111001111000",
                 "0011110000111100","0111100000011110","1111000000001111","1110000000000111",
                 "1100000000000011","1000000000000000","0000000000000000","1100000000000011",
                 "0011000000001100","0001100000011000","0000110000110000","1000011001100001"),
         1 =>   ("1000000000000000","1100000000000000","1110000000000000","1111000000000000",
                 "1111100000000000","1111110000000000","1111111000000000","1111111100000000",
                 "1111111110000000","1111111111000000","1111111111100000","1111111111110000",
                 "1111111111111000","1111111111111100","1111111111111110","1111111111111111"),
         2 =>   ("1111111111111111","1111111111111110","1111111111111100","1111111111111000",
                 "1111111111110000","1111111111100000","1111111111000000","1111111110000000",
                 "1111111100000000","1111111000000000","1111111111000000","1111100000000000",
                 "1111000000000000","1110000000000000","1100000000000000","1000000000000000"),
         3 =>   ("1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000",
                 "1111111111111111","0000000000000000","1111111111111111","0000000000000000"));

    CONSTANT all_birthdays  :   TEAMS_BIRTHDAYS(0 to 3) :=
        (0 =>   ("00100101","10011001","00000011","10011111"),
         1 =>   ("00000011","00100101","00000011","00001101"),
         2 =>   ("10011111","10011111","00000011","00000001"),
         3 =>   ("00000000","00000000","00000000","00000000"));
begin
    segment_CLK_DIVIDE : process (basysClock) is
    variable noOfBasyWaves  :   INTEGER := 0;
    constant maxCount       :   INTEGER := 50000;
    --constant maxCount        :   INTEGER := 32;
    begin
        if rising_edge(basysClock) then
            if noOfBasyWaves < maxCount then
```



```
                noOfBasyWaves := noOfBasyWaves + 1;
            else
                noOfBasyWaves := 0;
                segment_CLK <= not segment_CLK;
            end if;
        end if;
    end process;

    led_CLK_DIVIDE : process (basysClock) is
    variable noOfBasyWaves_LED  :   INTEGER := 0;
    constant maxCount_LED       :   INTEGER := 6250000;
    --constant maxCount_LED       :   INTEGER := 8;
    begin
        if rising_edge(basysClock) then
            if noOfBasyWaves_LED < maxCount_LED then
                noOfBasyWaves_LED := noOfBasyWaves_LED + 1;
            else
                noOfBasyWaves_LED := 0;
                led_CLK <= not led_CLK;
            end if;
        end if;
    end process;

    increment_segmentSequenceNumber : process(segment_CLK) is
    begin
        if rising_edge(segment_CLK) then
            -- seg_sequenceNumber must be converted to unsigned before undergoing arithmetic
            seg_sequenceNumber <= STD_LOGIC_VECTOR(unsigned(seg_sequenceNumber) + 1);
        end if;
```



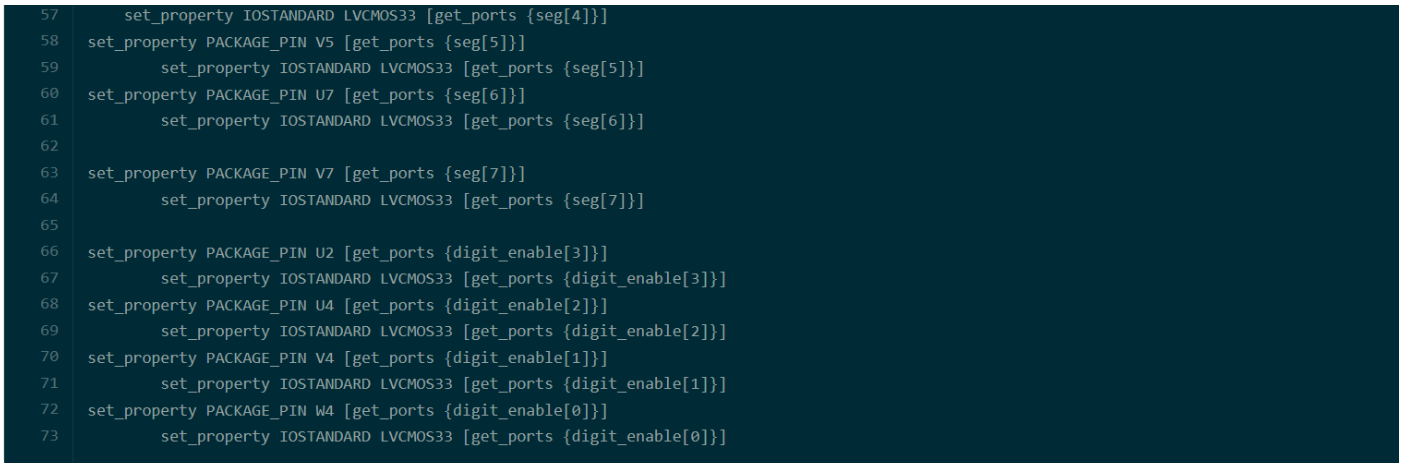
```
    end process;

    increment_ledSequenceNumber : process(led_CLK) is
    begin
        if rising_edge(led_CLK) then
            led_sequenceNumber <= STD_LOGIC_VECTOR(unsigned(led_sequenceNumber) + 1);
        end if;
    end process;

    controlDisplayEnables : process(seg_sequenceNumber) is
    begin
        case to_integer(unsigned(seg_sequenceNumber)) is
            when 0 =>
                digit_enable <= "0111";
            when 1 =>
                digit_enable <= "1011";
            when 2 =>
                digit_enable <= "1101";
            when others =>
                digit_enable <= "1110";
        end case;
    end process;

    personSelection : process (sw,seg_sequenceNumber,led_sequenceNumber) is
    VARIABLE selectPerson   :   INTEGER := 0;
    begin
        selectPerson := to_integer(unsigned(sw));

        seg <= all_birthdays(selectPerson)(to_integer(unsigned(seg_sequenceNumber)));
        led <= all_sequences(selectPerson)(to_integer(unsigned(led_sequenceNumber)));
    end process;

end Behavioral;
```

9.2 Simulation Source Code

```vhdl
----------------------------------------------------------------------------------
-- Company: 
-- Engineer: 
-- 
-- Create Date: 19.03.2018 16:29:07
-- Design Name: 
-- Module Name: seven_segment_TB - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------


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;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-----------------------------------------------------------------------
--------------------------entity declaration --
-----------------------------------------------------------------------
entity seven_segments_TB is
--  Port ( );
end seven_segments_TB;

-----------------------------------------------------------------------
----------------------------architecture-------------------------------
-----------------------------------------------------------------------

architecture arch of seven_segments_TB is

-----component declaration for UUT-------------------------------------

component seven_segments is
    port (basysClock    :   IN STD_LOGIC;
        sw            :   IN STD_LOGIC_VECTOR(0 to 1);
        led           :   OUT STD_LOGIC_VECTOR(0 to 15);
        seg           :   OUT STD_LOGIC_VECTOR(0 to 7);
        digit_enable  :   OUT STD_LOGIC_VECTOR(0 to 3));
    end component;

-----signals required to test design-----------------------------------

signal basysClock : std_logic;
signal sw : std_logic_vector(0 to 1);
signal led : std_logic_vector(0 to 15);
signal seg : std_logic_vector(0 to 7);
signal digit_enable : std_logic_vector(0 to 3);
-----------------------------------------------------------------------

-----Period of the Basys-3 board clock---------------------------------
constant T_clk : time := 10 ns;
-----Period used for the simulation------------------------------------
constant sim_clk : time := 100 * T_clk;
-----The number of clock cycles to run through-------------------------
constant n_cycles : integer := 20;
-----------------------------------------------------------------------
-----------------------------------------------------------------------

begin

-----instantiation of seven_segments-----------------------------------
-----------------------------------------------------------------------
  DUT : seven_segments
  port map( basysClock => basysClock,
            sw => sw,
            led => led,
            seg => seg,
            digit_enable => digit_enable);

-----clock generator process-------------------------------------------
-----------------------------------------------------------------------

  basys_clk_gen : process is

  begin

   while now <= (n_cycles*sim_clk) loop

           basysClock <= '1'; wait for T_clk/2; --alternating clock pulses
           basysClock <= '0'; wait for T_clk/2;

         end loop;

  wait;

  end process;
 ----------------------------------------------------------------------
 ----------------------------------------------------------------------

 -----stimulus process; The only stimuli are the two switches----------
 ----------------------------------------------------------------------
  stimulus : process is

  begin

   sw(0) <= '0'; sw(1) <= '0'; wait for 5*sim_clk; --Switch setting "00"
   sw(0) <= '0'; sw(1) <= '1'; wait for 5*sim_clk; --Switch setting "01"
   sw(0) <= '1'; sw(1) <= '0'; wait for 5*sim_clk; --Switch setting "10"
   sw(0) <= '1'; sw(1) <= '1'; wait for 5*sim_clk; --Switch setting "11"
   wait;

  end process;
 ----------------------------------------------------------------------
 ----------------------------------------------------------------------
end arch;
```

9.3 Constraints Source Code

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports basysClock]
set_property IOSTANDARD LVCMOS33 [get_ports basysClock]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports basysClock]

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]

## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
```



```
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN V7 [get_ports {seg[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {seg[7]}]

set_property PACKAGE_PIN U2 [get_ports {digit_enable[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {digit_enable[3]}]
set_property PACKAGE_PIN U4 [get_ports {digit_enable[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {digit_enable[2]}]
set_property PACKAGE_PIN V4 [get_ports {digit_enable[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {digit_enable[1]}]
set_property PACKAGE_PIN W4 [get_ports {digit_enable[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {digit_enable[0]}]
```