**COL215P**
**Assignment 1: Stopwatch**

**Shreya Arora(2019CS10401)**
**Ishita Chaudhary (2019CS10360)**

The implementation for constructing a stopwatch using LED displays and push buttons on the Basys 3 board consists of the following five modules:

1. Seven-segment decoder for a single LED display (seven_segment_decoder.vhdl)
2. Timing circuit to drive all LED displays (multiplexer in the main file)
3. Modulo N Counter :one for stopwatch (clockdivider.vhdl) and one for digit display (clk_refresh.vhdl)
4. Working of the Stopwatch (the main file: stopwatch.vhdl)
5. Debouncing for push button (debounce.vhdl)

## Module 1: Seven-segment decoder for a single LED display,

In order to implement a seven-segment decoder for a single LED display, we first created a truth table for the four anode_signal input variables (A) and the corresponding seven cathode_signal output variables (C) for the digits 0-9. The truth table is displayed as follows:

| Num | A[3] | A[2] | A[1] | A[0] | C[6] (A) | C[5] (B) | C[4] (C) | C[3] (D) | C[2] (E) | C[1] (F) | C[0] (G) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **1** | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| **2** | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **4** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| **5** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **6** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **8** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Truth Table

Using the K-Maps as shown above, we developed the minimized combinational logic for the 7 output variables which are given as follows:

(‘.’ is equivalent to the AND operator)
(‘+’ is equivalent to the OR operator)
(’ is equivalent to the NOT operator)

To arrive at the minimized combinational logic for the seven-segment display output signals, we minimized the following functions using K-maps.

1. C(0) = Σm(2,3,4,5,6,8,9)
2. C(1) = Σm(0,4,5,6,8,9)
3. C(2) = Σm(0,2,6,8)
4. C(3) = Σm(0,2,3,5,6,8,9)
5. C(4) = Σm(0,1,3,4,5,6,7,8,9)
6. C(5) = Σm(0,1,2,3,4,7,8,9)
7. C(6) = Σm(0,2,3,5,6,7,8,9)

**K-Maps to arrive at the Minimized Combinational Logic**

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 1 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(0):** C(0) = Σm(2,3,4,5,6,8,9)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 0 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(1):** C(1) = Σm(0,4,5,6,8,9)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 1 |
| 01 | 0 | 0 | 0 | 1 |
| 11 | x | x | x | x |

| | | | | |
|---|---|---|---|---|
| 10 | 1 | 0 | x | x |

**K-Map for C(2):** C(2) = Σm(0,2,6,8)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(3):** C(3) = Σm(0,2,3,5,6,8,9)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(4):** C(4) = Σm(0,1,3,4,5,6,7,8,9)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | 1 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(5):** C(5) = Σm(0,1,2,3,4,7,8,9)

| A(1) A(0) / A(3) A(2) | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| 11 | x | x | x | x |
| 10 | 1 | 1 | x | x |

**K-Map for C(6):** C(6) = Σm(0,2,3,5,6,7,8,9)

**Minimized Combinational Logic for Seven Segment Display Output Signals**

1. **C(0)** <= not ((A'(1).A'(3)) + (A'(0).A(2)) + (A'(0).A(1).A(3)) + (A(0).A'(1).A'(2)) + (A(0). A'(3)) + (A(1).A(2)))
2. **C(1)** <= not ((A'(0).A'(2).A'(3)) + (A'(0).A(2).A(3)) + (A'(1).A'(3)) + (A(0).A'(2).A(3)) + (A'(0).A'(1)))
3. **C(2)** <= not((A'(0).A(1)) + (A(0).not A(1)) or (not A(2).A(3)) or (not A(0).not A(2)) + (not A(0).A(3)))
4. **C(3)** <= not((A(1).A'(2).A(3)) + (A'(0).not A(1).A'(3)) + (A'(1).A(2).A(3)) + (A(1).A(2).A'(3)) + (A(0).A'(2)))
5. **C(4)** <= not((A'(1).A'(3)) + (A(2).A'(3)) + (A(0).A(2)) + (A(0).A(1)))
6. **C(5)** <= not((A'(2).A'(3)) + (A'(0).A(1).A'(2)) + (A(1).A'(3)) + (A(0).A'(1)) + (A(0).A(2)))
7. **C(6)** <= not((A'(1).A(2)) + (A(0).A'(1)) + (A'(0).A(1).A'(2)) + (A(2).A'(3)) + (A(0).A(3)))

**Module 2: Timing circuit to drive all LED displays**

The multiplexer module has primarily 2 inputs: 1) The clock signal and 2) The time_count signal. The output signal affected by the multiplexer module are:

- The **anode_signal**: **four 4-bit inputs** where the index of the anode input having 0 value denoted the index of the seven segment display number being illuminated
- The **decimal**: **boolean inputs** specifying if a decimal point should appear before the displayed number on the seven segment display.
- The **display_number**: for assigning the suitable time value (minute, second_tens, second_ones, tenth_second) to be displayed on the seven-segment display.
- The **time_count**: was incremented by 1

**Case 1:** In the presence of a rising edge, when the time_count is 11, the multiplexer will drive the anode signal to display the minute digit on the seven-segment display board, anode_signal will be assigned the value 0111. Since we do not need a decimal point before the displayed digit, decimal = 0, and time_count will be incremented and assigned a value of 00.

**Case 2:** In the presence of a rising edge, when the time_count is 00, the multiplexer will drive the anode signal to display the tens digit of the seconds' value (second_tens) on the seven-segment display board, anode_signal will be assigned the value 1011. In order to separate the minute value and the second value, we will need a decimal point before the displayed digit, decimal = 1, and time_count will be incremented and assigned a value of 01.

**Case 3:** In the presence of a rising edge, when the time_count is 01, the multiplexer will drive the anode signal to display the ones digit of the seconds' value (second_ones) on the seven-segment display board, anode_signal will be assigned the value 1101. Since we do

not need a decimal point before the displayed digit, decimal = 0, and time_count will be incremented and assigned a value of 10.

**Case 4:** In the presence of a rising edge, when the time_count is 10, the multiplexer will drive the anode signal to display the tenth-second digit (tenth_second) on the seven-segment display board, and anode_signal will be assigned the value 1110. In order to separate the second value and the tenth-second value, we will need a decimal point before the displayed digit, decimal = 1, and time_count will be incremented and assigned a value of 11.

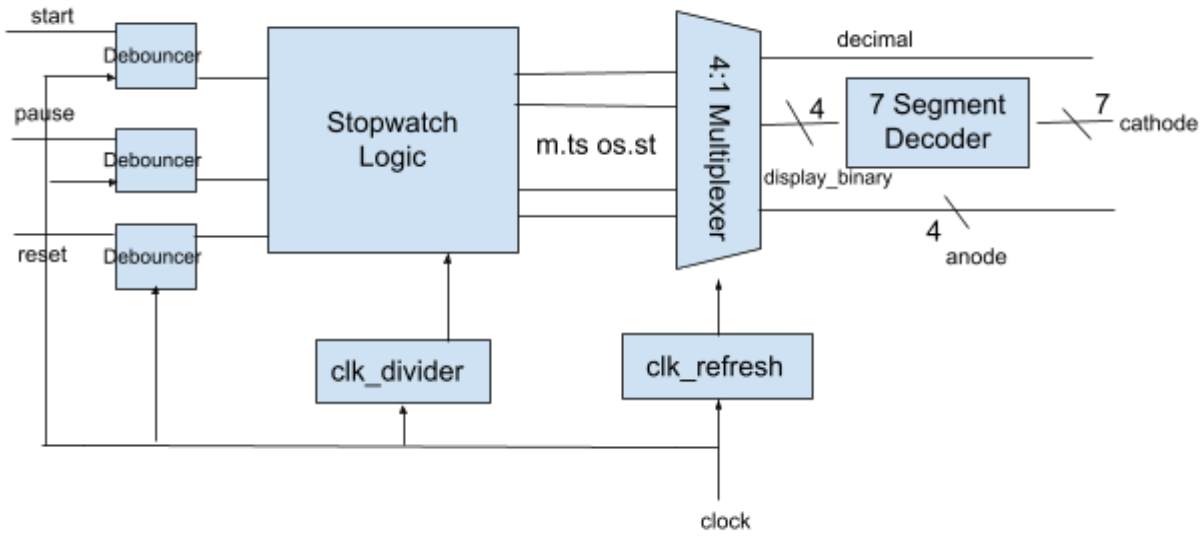
### Module 3: Modulo N Counter

The frequency of the clock on the BASYS board is 100Mhz (10^8 Hz). The purpose of one of the clock dividers is to reduce this frequency to make it human-perceivable. The refresh period should be within 1 to 16 ms, implying the digit period to be between 0.25 ms and 4 ms.

Clk_refresh frequency => 10^8/10^5 = 1000 Hz = 1ms digit period
So we see a rising edge of clk_refresh every 4 ms.
The other clock divider with output frequency = 10Hz is used to implement the stopwatch. 10Hz corresponds to a period of 0.1s, the lowest time unit (the tenth of a second) we display on the stopwatch.

We have implemented two clock divider modules to divide the clock's frequency from 100 MHz to 10 Hz (for stopwatch) and 1000 Hz (=1 ms digit period) to have a suitable display frequency. For this, we use modulo N counter with N = $10^7$ & N = $10^5$ respectively. We have an input clock signal and return an output signal with reduced frequency. We start with a time counter equal to 0, and at a rising edge of the clock, the time counter will get incremented by 1. If the time counter value is equal to greater than N, then its value is reset to 0, else it stays at the same value.

### Module 4: Working of the Stopwatch



The ensemble is driven by a 10 Hz timing reference. We provide an enable input and a reset input. The enable input comes from a flip-flop/latch, which is set to ‘1’ when the Start/Continue button is pressed and set to ‘0’ when the Pause button is pressed. Reset input comes from a push button. The main file calls all other procedures. Above is the block diagram of the stopwatch.

The stopwatch ranges from 0:00:0 to 9:59:9, i.e., 10 minutes.
Push buttons are as follows:
The design consists of 3 push buttons that work the same as any other stopwatch.
● Start/Continue (middle button)
● Pause (left button)
● Reset (top button) The reset button pauses the stopwatch and sets it to 0:00:0.

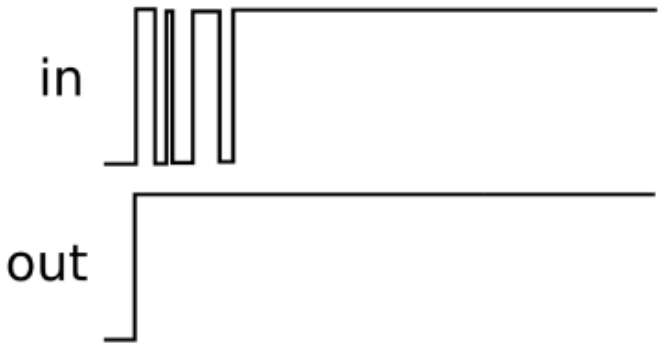
The signals m (minutes), s1(tens place of seconds), s2 (units place of seconds) and ms (deci-seconds/ tenth of second) change as per the 10Hz clock (clk10hz) made by dividing the in-built 100MHz clock. These signals are then converted into corresponding seven-segment cathode values. Then, the four digits are displayed by setting the anodes as per clk_refresh (1000 Hz clock) so that the user will perceive all four digits together. Two decimal pointers are also displayed using the “dp” bit to represent the time as m.s1s2.ms

The code below shows the stopwatch logic depicted in the block diagram.

```
                if enable_watch = '1' then

-- Modulo counter for tenths of a second

                    if tenth_second<9 then
                        temp_tenth_second <= tenth_second + 1;
                    elsif tenth_second=9  then
                        temp_tenth_second <= 0;
                    end if;

-- Modulo counter for ones digit of seconds

                    if tenth_second=9 then
                        if second_ones<9 then
                            temp_second_ones <= second_ones + 1;
                        elsif second_ones=9 then
                            temp_second_ones <= 0;
                        end if;
                    end if;

-- Modulo counter for tens digit of seconds

                    if (tenth_second=9) and (second_ones=9) then
                        if second_tens<5 then
                            temp_second_tens <= second_tens + 1;
                        elsif second_tens=5  then
                            temp_second_tens <= 0;
                        end if;
                    end if;

-- Modulo counter for minutes

                    if (tenth_second=9) and (second_ones=9) and (second_tens=5) then
                        if minute<9 then
                            temp_minute <= minute + 1;
                        elsif minute=9  then
                            temp_minute <= 0;
                        end if;
                    end if;



               end if;

            end if;
```

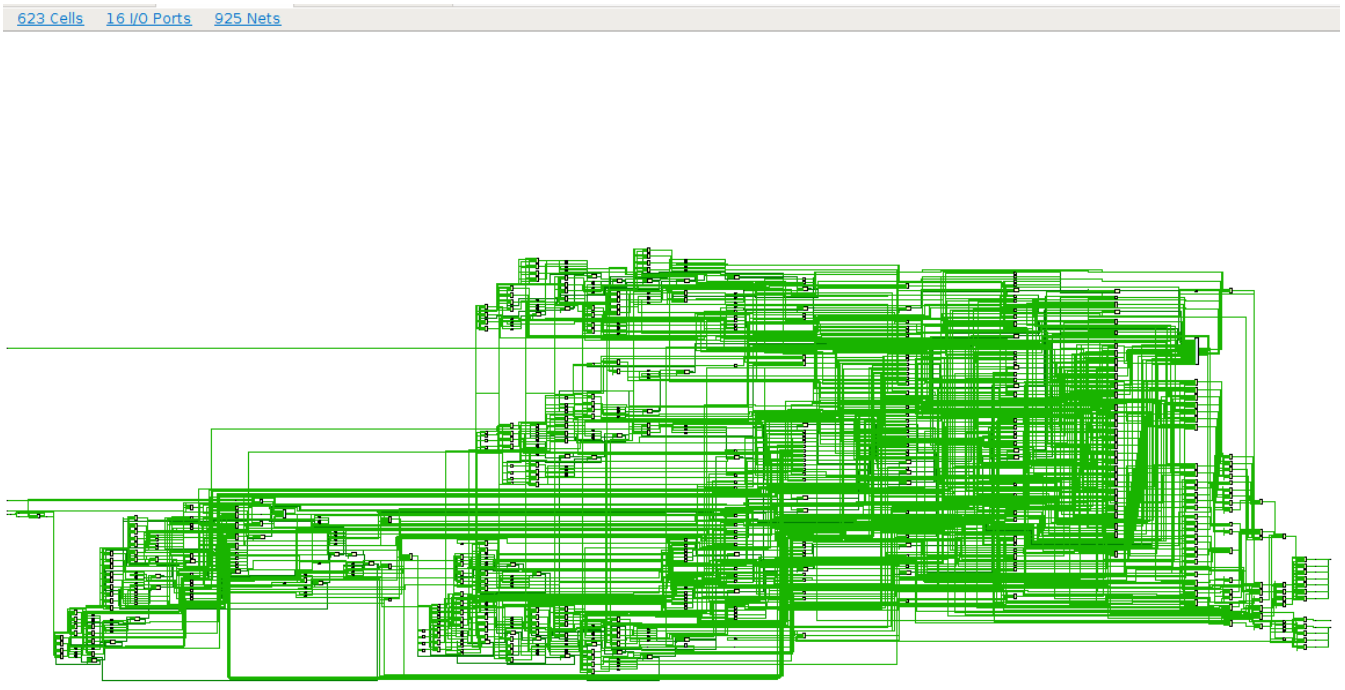
**Module 5: Debouncing for push button**

The entity is needed to stabilize the input signal received from the push buttons. This is done by taking readings of the button after fixed time intervals. This ensures that the noise is removed.




Debouncer's Input & Output

**Synthesized Design & Utilization Stats by Vivado**




The above screenshot shows the design uses 623 cells, 16 I/O ports, and 925 Nets. Following the truth table above, we have built a combinational circuit using k-maps for all seven segments. We then wrote the constraints file to specify the switches, cathodes, and anodes.

Next, we perform a synthesis of our design, followed by implementation. We then generate the bitstream of the implemented design, which will finally be programmed into the FPGA device.

**Utilization** - Post-Implementation

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 289 | 20800 | 1.39 |
| FF | 331 | 41600 | 0.80 |
| IO | 16 | 106 | 15.09 |
| BUFG | 5 | 32 | 15.63 |

**Testing**

Attached are the photos of our stopwatch vs. the stopwatch on the phone. The constant difference in the order of deci-seconds is due to the initial difference in pressing the start button.