Management and analysis of physics dataset
HW1: FPGA Stopwatch (modulus 16)

Rocco Ardino	Alessandro Lambertini	Alice Pagano	Michele Puppin
1231629	1242885	1236916	1227474

Wednesday 29th April, 2020

1 Aim

The purpose of the assignment is to implement a 4-bit stopwatch, namely a counter, with the following functionalities:

- **START**: it enables the counting.
- **STOP**: it stops the counting.
- **RESET**: it resets the counting.
- **FREQUENCY SELECTORs**: they can change the frequency of the counting.
- **REVERSE SELECTOR**: it makes the stopwatch counting in reverse.

2 Implementation

The Arty7 board has 4 LEDs that could be used as a display counter ($0 \rightarrow 15$). Indeed, each LED is associated to a bit: an off LED corresponds to the bit state '0', while a bright one corresponds to the bit state '1'. The time flow is regulated by the embedded clock of the board, which is used to implement the counter. Concerning the functionalities, START, STOP and RESET can be triggered by the embedded buttons of the board, while FREQUENCY and REVERSE SELECTORs by the four switches. In particular, three switches are used for modulating the frequency of the counting and one is used for reversing it. The actual disposition of functions is illustrated in Figure 1.

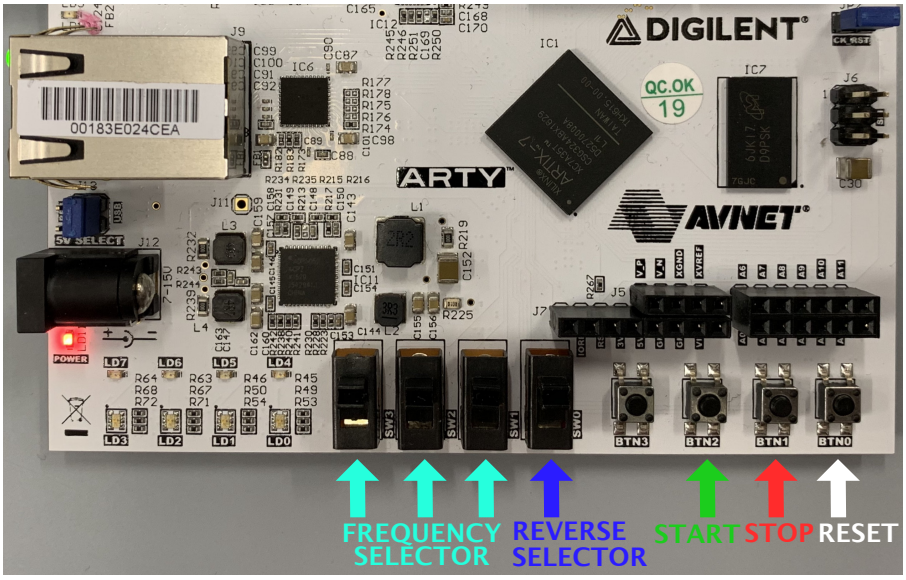

Figure 1: Arty7 board: disposition of the functionalities.

2.1 Counter

The code implementation is constituted by four main processes. The first two processes (`p_cnt` and `p_slw_cnt`) are used to implement the counter.

First of all the clock has been used to increase the value of a vector of 28 bits (`counter`) each time the clock signal shows a rising edge, as in process `p_cnt` (Listing 1).

```
p_cnt : process(clk,rst,sel_in) is
    begin
    if rst = '1' then
        counter <= (others => '0');
    end if;
    if rising_edge(clk) then
        counter <= counter +1;
    end if;
end process;
```

Listing 1: `p_cnt` process.

Then, since the speed of the embedded clock is too fast, it has been slowed down in the process `p_swl_cnt` (Listing 2) by creating a slow clock signal `slow_clk` and taking the i^{th} bit `slow_clk <= counter(i)`, where the value of i is determined by the frequency selector. The slow clock signal has eventually been used to update the `slow_counter` signal, which reflects the four bit display counter that will be mapped to the LEDs.

```
p_slw_cnt : process(clk,rst,frz,slow_clk,sel_in,state) is
    begin
    if rst = '1' then
        slow_counter <= (others => '0');
    end if;
    if rising_edge(clk) then
        slow_clk_p <= slow_clk;
        if state = '0' then
            if slow_clk = '1' and slow_clk_p = '0' then
                if sel_in(0) = '0' then
                    slow_counter <= slow_counter + 1;
                elsif sel_in(0) = '1' then
                    slow_counter <= slow_counter - 1;
                end if;
            end if;
        end if;
    end if;
end process;
```

Listing 2: `p_slw_cnt` process.

2.2 FREQUENCY and REVERSE SELECTOR

In order to select manually the speed of the display counter, the value of the i^{th} element of the vector `counter` is chosen based on the values of the three elements of `sel` vector. The ladder reflects the status of the three switches. It is implemented in the `speed` process (Listing 3). Moreover, in order to reverse the counter, `slow_counter` is updated forward ($+1$) or backward (-1) based on the position of a fourth switch as shown in process `p_slw_cnt` (Listing 2).

```
speed : process(clk,rst,slow_clk,sel) is
    begin
    case sel is
        when "000" => slow_clk <= counter(27);
        when "001" => slow_clk <= counter(26);
        when "010" => slow_clk <= counter(25);
        when "011" => slow_clk <= counter(24);
        when "100" => slow_clk <= counter(23);
        when "101" => slow_clk <= counter(22);
        when "110" => slow_clk <= counter(21);
        when "111" => slow_clk <= counter(20);
```

```
        when others => null;
    end case;
end process;
```

Listing 3: `speed` process.

2.3 START, STOP and RESET

In details when START button is pressed and then released the board starts counting, when STOP button is pressed and released the counter stops and the LEDs freeze. Finally, when RESET button is pressed and released, the counter goes to zero and stops at the same time.

START and STOP button have been implemented in the process `btn_state` (Listing 4), where a variable `state` '1' if STOP button is pressed, while it is set to '0' if START is pressed. In the process `p_slw_cnt`, previously described in Listing 2, the update of `slow_counter` happens only if `state` is set to '0'. Instead, RESET button, besides setting `state` to '1' in order to stop the counter, sets `counter` and `slow_counter` to '0'.

```
btn_state : process(clk,rst,frz,start,slow_clk,sel_in,state) is
    begin
    if rising_edge(clk) then
        if start = '1' then
            state <= '0';
        elsif frz = '1' then
            state <= '1';
        elsif rst = '1' then
            state <= '1';
        end if;
    end if;
end process;
```

Listing 4: `btn_state` process.

3 Simulation

To be sure of the correct behaviour of the counter, we performed a behavioural simulation on Vivado, where the unit under test is `cnt`, in Listing 5.

```
uut : cnt port map (clk       => clk,
                    sel_in(0) => rev,
                    sel_in(1) => sel(0),
                    sel_in(2) => sel(1),
                    sel_in(3) => sel(2),
                    y_out     => led_out,
                    frz       => btn_frz,
                    start     => btn_start,
                    rst       => btn_rst);
```

Listing 5: `uut` process.

First of all, we fixed the simulation time to 10 μs. To visualize the behaviour of the counter in that time scale, we set a higher frequency of the counter for the whole time of the simulation. Then we defined a clock signal consisting in a square wave of period 10 ns, whose transitions are implemented in `p_clk` process. These two processes are illustrated in Listing 6.

```
p_clk: process
begin
    clk <= '0'; wait for 5 ns;
    clk <= '1'; wait for 5 ns;
end process;
end process;
```

Listing 6: `speed` and `p_clk` process.

Afterwards, we simulated the following routine in `p_start` process (Listing 7):

- the RESET button is pressed for 50 ns and then released;
- after 50 ns START button is pressed for 50 ns and released;
- after 1200 ns the STOP button is pressed and released after 50 ns.

```
p_start: process
begin
    btn_rst   <= '1'; wait for 50 ns;
    btn_rst   <= '0'; wait for 50 ns;
    btn_start <= '1'; wait for 50 ns;
    btn_start <= '0'; wait for 1200 ns;
    btn_frz   <= '1'; wait for 50 ns;
    btn_frz   <= '0'; wait for 50 ns;
end process;
```

Listing 7: `p_start` process.

In Figure 2 the results of the simulation are shown. The signal `led_out` carries the value of the counter at a certain time. By this way we can see that the counter is working correctly, as well as the START, STOP and RESET functionalities.

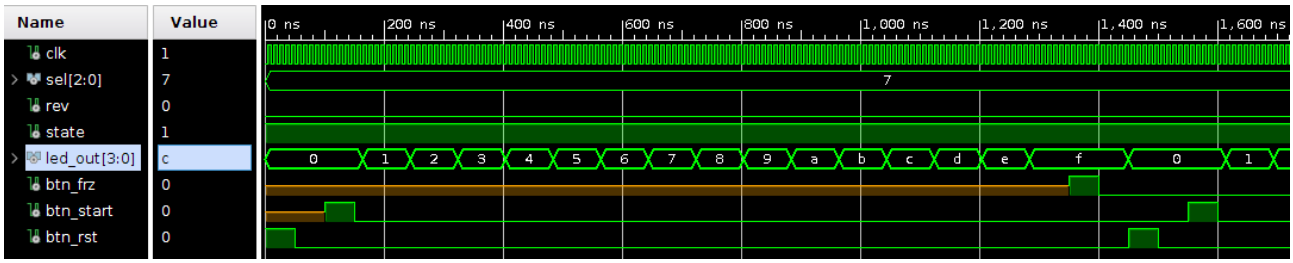

Figure 2: Vivado simulation of FPGA Stopwatch.

4 Conclusion

This assignment presents the design of a FPGA-based Stopwatch with START, STOP and RESET functionalities. A behavioral validation testbench has been developed and has been used to confirm a proper behaviour without glitches. Afterwards, the system has been experimentally validated on Arty7 board.