Design and implementation of a stopwatch on FPGA

Segalini Beatrice (1234430) and Iriarte Delfina (1231682)

October 23, 2020

1 Introduction

A stopwatch is a time-keeping device that measure the time elapsed from the start to end of any events. In this report, a digital stopwatch is modeled using **VHDL** and implemented in the *Arty 7* board. It is controlled by three input buttons, namely:

- START: enable counting;
- STOP: disable counting and freeze the LED;
- RESET: disable counting, reset to zero.

The output is given by 4 LEDs that light up, representing the seconds in the time range $[0, 15]$s in binary notation.

2 Architecture and design of the stopwatch in VHDL

In this section, the design of a stopwatch is analysed. The inputs consist of the three already mentioned buttons (START, STOP and RESET) and the internal clock of the *Arty 7* board, which has a built-in 100 MHz oscillator. The buttons can be manually triggered on the *Arty 7* board, which also is used to display the output as a 4-bit LEDs.

In order to build the stopwatch, one first needs to define a counter process like the one presented in **Listing 1**. When the reset button is pressed the counter is set to 0, if not the counter increments at each rising edge of the internal clock; finally, when it reaches its maximum value, it rolls over to 0 and repeats.

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

Listing 1: Basic counter process for the implementation of the stopwatch.

However, the `p_cnt` process is not enough to produce the desired output. In fact, the main clock frequency applied to the module is 100 MHz, meaning that a cycle lasts only 10 nanoseconds. Since the stopwatch needs to show times of a completely different order of magnitude,

slowing the process down is compulsory in order to make the output signal perceivable for human eyes: hence, a `slow_counter` is defined and obtained by taking the 27^{th} bit of the counter (`cnt(26)`). In this way, the LEDs will switch every 10ns $\times$ 2^{27} = 1.34s since the clock cycle time is 10 ns and each position of the `cnt` vector represents a multiplicative factor 2.

After this remark, it is possible to define the `p_slow_cnt` process, reported in **Listing 2**, where the implementation of the START, RESET and STOP buttons are described.

Similarly to 1, the RESET button action is to bring to 0 the value of the `slow_counter`. To properly characterize the effect of pressing START and STOP buttons, a support variable ss1 is then introduced. The support variable represents the signal that allows the START button: only when it is equal to 1, the `slow_counter` is updated. Thus, once the START button is pressed, the counter is enabled whereas if the STOP button is on, the counter stop and the LEDs freeze.

```
p_slw_cnt: process(clk, rst, start, stop, slow_clk) is

    begin
        if rst = '1' then
        slow_counter <= (others => '0');

        elsif rising_edge(clk) then
            if start = '1' and stop ='0' then
                ss1 <= '1' ;
            elsif stop = '1' and start = '0' then
                ss1 <= '0' ;
            end if ;

            if ss1 = '1' then
                slow_clk_p <= slow_clk;
                if slow_clk ='1' and slow_clk_p = '0' then
                    slow_counter <= slow_counter + 1;
                end if;
             end if;

             if ss1 = '0' then
                slow_clk_p <= slow_clk;
                if slow_clk ='1' and slow_clk_p = '0' then
                    slow_counter <= slow_counter;
                end if;
            end if ;

     end if;
end process;
y_out <= std_logic_vector(slow_counter);
```

Listing 2: Slow counter process and buttons coding.

The implementation on the *Arty 7* board is performed by using the constraints that allows to map an input or output signal to the appropriate pin, such as a LED or a button. The synthesis, implementation, and generation of *bitstream* file are required as final steps to physically create the stopwatch.

3 Simulation

The blinking frequency that was slowed down for the physical implementation (1 LED blink every $\approx 1\,\mathrm{s}$) proved to be too slow to actually visualize properly the simulation results. As a consequence, in the simulation phase, the counter frequency was reset to its original values.

The simulation testbench consists mainly of two processes: one of them corresponding to the clock generation and the other one is the stimuli process, composed of the definition of the action of the main signals, i.e. the buttons.

In Figure 3 the results of the waveform simulation are shown. As expected, it can be seen that when the START button is pressed, `y_out` (the output) starts to increment its value. It is also worth noticing that when the STOP button is pressed, the output value `y_out` is frozen, while the RESET button makes the counter reset.

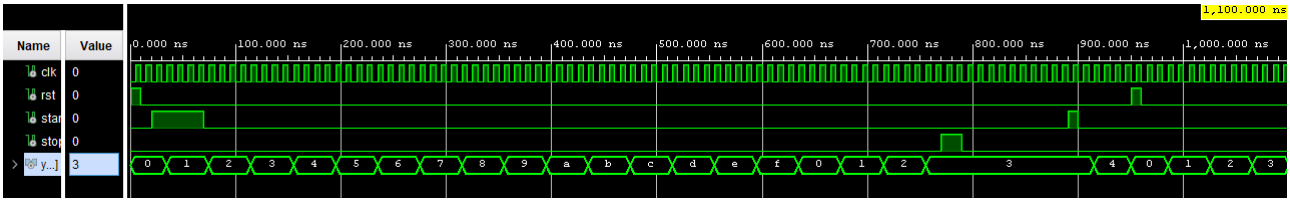

Figure 3: Simulation results: one can observe the first phase in which the stopwatch is changing its counter value from 0 to 15 and then restarting; then the action of the STOP button can be seen; finally also the effect of START and RESET buttons are tested.

The simulation results are consistent and proved that the stopwatch is correctly programmed and implemented.

4 Conclusion

In this work, a digital stopwatch with STOP, START and RESET buttons has been designed using **VHDL** and implemented successfully on the *Arty 7* board. Furthermore, the behavior of the system was verified by developing a behavioral validation testbench, whose results are proof of good design and implementation.