Settable countdown timer with 4-digit 7-segment display

Submitted as part of the requirements for:

CE869 High Level Digital Design

Name: Lorenzo Paris (1700982)

Lecturer: Luca Citi

Date: 15 February 2018

Number of words: 2095

Table of Contents

1 Assignment Introduction

For this assignment I have been asked to implement a settable countdown timer by using VHDL code, the Xilinx Vivado software and the Basys3 design board [1].
The whole project has been designed to be implemented gradually by three different tasks in order to:

- make the students expertise in such implementations;
- have a guide line for implementing harder structures.

The report will depict, in the following sections, the development of this assignment task by task by highlighting the main issues occurred during the design, the ideas arose in the planning stage and the explanation of every choice made during the implementation phase.

2 Task 3

2.1 Task Introduction

The final task, ***Task 3***, will allow the user to make use of the 5 buttons of the Basys3 board to simulate a countdown timer. Particularly, the central button allows the user to toggle between:

- *SET MODE*: pause and set the countdown timer;
- *GO MODE*: start the countdown time at a rate of approximately 1 second.

The other four buttons have the purpose to increment, during *SET MODE*, from 0 to 9 each of the four digits displayed on the LED display of the board as follows:

- **Right button**: increments the seconds;
- **Down button**: increments the tens of seconds;
- **Up button**: increments the minutes;
- **Left button**: increment the tens of minutes.

The top-level entity, moreover, must make use of the previously implemented ***four_digits*** and ***one_digit*** entities.

2.2 Planning

In order to achieve a proper behaviour by the board, some important aspects need to be considered.
To begin with, the 4 anodes of the seven LEDs are tied together into one "common anode", but the LED cathodes remain separate [2]. Therefore, it is not possible to illuminate all the four digits at the same time. The idea, thus, is to show each digit at rate frequency which is not perceivable by human eye. For this reason, I will make use of a frequency rate of 500 MHz by which every digit will be displayed in turn.
Regarding the countdown timer implementation, my first idea was to split the *SET MODE* and *GO MODE* functionalities into two different entities. In fact, I firstly created an entity called ***timer_set***, which allowed to increment each digit on the 7-segment display. Once it worked properly, I created the other entity called ***timer_go***, which decreased, at a rate of 1 second, a default set time. The problem arose when I had to wire, at the same time, both of the entities with the top-level entity. The main error related to this scenario, was to try to wire the same signal containing the time counter for being incremented in ***timer_set*** and decreased in ***timer_go***. Obviously, the two behaviours should have not occurred together because toggled by the central button; however, VHDL does not allow to alter the value of a single signal in different processes.
Finally, I decided to encapsulate both of the behaviours (*SET* end *GO*) inside the same entity, which I called ***timer***. Therefore, by having a single process, I was able to share and update the time counter between the two functionalities.
The final code is divided into 4 different VHDL entities, hierarchically structured as follows:

- ***one_digit***: is the bottom-level entity;
- ***four_digits***: it has as dependencies the ***one_digit*** entity;
- ***timer***: it has as dependencies the ***four_digits*** entity, followed by the ***one_digit*** entity
- ***main3_final***: is the top-level entity which has as dependencies the ***timer*** entity, followed by ***four_digits*** and ***one_digit*** entity.

2.3 Implementation

2.3.1 main3_final Entity

In order to implement the desired behaviour, this entity needs as input all the five buttons jointly with the master clock, which, in the Basys3 board, provide a frequency rate of 100 MHz. Furthermore, it needs, as output, a 7 bit vector for displaying the 7 segments which will be later decoded inside the ***one_digit*** entity, the 4 anode signals computed in the ***four_digits*** entity and the point, implemented in the ***timer*** entity, which will be only useful for the user to discretyze the two timer functionalities.
The only one behaviour related to the top-level entity, is to divide the master clock frequency into two sub-frequencies much more useful for our purpose:

- ***clk_1Hz***: with a period of (approximately) 1 second, is used to simulate the countdown timer;

- ***clk_500Hz***: with a frequency rate of (approximately) 500 Hz, allows the user to not perceive the segment LEDs to be showed in turn.

In order to generate the aforementioned sub-frequencies [3] I created a single process with a double frequency divider structured as follows:

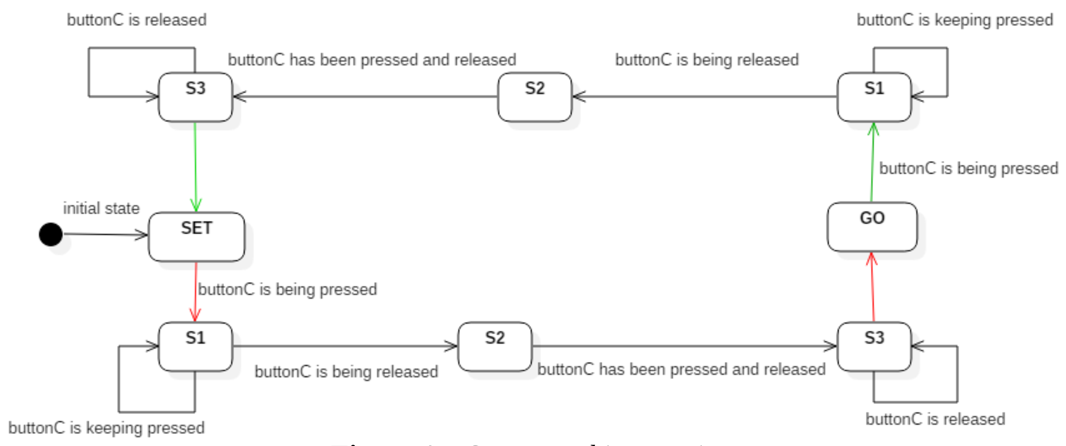
- A counter of 18 bit for the ***clk_500Hz*** will count up to a value of 200,000;
- A counter of 27 bit for the ***clk_1Hz*** will count up to a value of 100,000,000;
- The two counters increase by one every 200,000 rising edge of the master clock the ***clk_500Hz*** counter, and every 100,000,000 the ***clk_1Hz*** counter;

By doing this, the most significant bit of each counter will be 1 once the respective limit value has been reached by the counting process.
Finally, the IN/OUT signals are then wired to the lower-level entity by using the *port map* keyword.

2.3.2 timer Entity

Instead of generating a timer counter for each digit, my idea was to create a single array of 16 bit, ***time_left***, in which every 4 bits represent a digit; therefore, by going from the rightmost 4 bits towards the leftmost 4 bits of the array, there will be a 4 bits representation for seconds, tens of seconds, minutes and tens of minutes respectively.

State Machine Timer Mode
In order to make the central button changes the timer mode, I created a state machine with 2 states named ***set*** and ***go***. A process, called ***state_machine_map*** will simply set the next state that the current state variable must follow. Another process, ***state_machine_engine***, will act as engine of the state machine based on the central button pressing event. Moreover, I created a further state machine with 3 states which allow the ***state_machine_engine*** to make the ***set*** and ***go*** states change only after having released the button itself, as depicted in the following diagram:



buttonC is released
buttonC has been pressed and released
buttonC is being released
buttonC is keeping pressed
S3
S2
S1
buttonC is being pressed
initial state
SET
GO
buttonC is being pressed
S1
S2
S3
buttonC is being released
buttonC has been pressed and released
buttonC is keeping pressed
buttonC is released

Figure 1 – *State_machine_engine*

Update Timer Process
Both of the timer behaviours are computed inside a single process called ***update_timer*** wich has the ***current_state*** signal within the sensitivity list. During this process, a series of nested IF/ELSE conditions update properly the ***time_left*** counter. A first IF statement is responsible of splitting the two timer modus operandi based on the current state of the state machine. Therefore, if the current state is in *SET MODE*, at every rising edge of the master clock will be checked if the user has pressed one of

the 4 buttons for incrementing ***time_left***; within every button checker, another nested IF statement has been included in order to avoid random incrementing behaviours; this IF section, in fact, allows the user a button pressing event of around 0.25 seconds; therefore, every 0.25 seconds the button is pressed, the timer increments by one. Conversely, if the current state is in *GO MODE*, at every rising edge of the 1 Hz clock (every second), ***time_left*** is decreased based on the current value of each digit.
In order to make the code more human readable, several conversions has been made among data types; therefore in the set mode IF statement, for checking out if a digit counter has reached a certain value, it is converted from binary to decimal representation; another example of conversion can be seen for decreasing/increasing a digit counter: it is converted first from binary to decimal, incremented/decreased by one, and finally converted again into binary.
The last part of the ***timer*** entity is represented by the port map keyword which wires the respective 4 bits of each digit of ***time_left*** with the digits decoded and displayed in the lower-level entities. Particularly, the ***four_digits*** entity will receive as input the clock at rate of 500 Hz which will be used to illuminate, in turn, every digit on the display.

2.3.3 four_digits Entity

In order to change the digit illuminated, I created a state machine with 4 states representing a single digit displayed in turn. In a process called ***state_machine_map***, aside from indicating the next state, I set the anode variable with the respective value representing the digit to be displayed in turn. The states change with a frequency rate of 500 Hz: this value is big enough to make the four digits appear bright and continuously illuminated [2].
Moreover, each digit is driven on a particular signal which will be captured and decoded by the ***one_digit*** entity.

2.3.4 one_digit Entity

The ***one_digit*** entity simply takes as input the binary digit to be decoded and converted into a combination of segment illuminated on the 7-segment display.
Therefore, this entity consists of a single process which will set the right 7-segment bit combination for representing a certain binary digit.

3 Task 2

3.1 Task Introduction

In this task I have been asked to allow the user to make use of the 16 switches of the board to represent 4 digits encoded by each group of 4 switches. The user, further-

more, manually simulates the multiplexing clock by pressing the central button; therefore, only one digit a time will be illuminated on the 7-segment display.
I have been provided of the top-level entity which must make use of the ***four_digits*** entity, which I had to design, and the ***one_digit*** entity previously developed in *Task 1*.

3.2 Planning

The first idea for implementing such a scenario was to create a simple state machine endorsed of 4 states. They would have changed at each clock signal event, that in this case is manually simulated by the central button pressing event. Since I was novice in designing sequential circuits in VHDL, my difficulties in developing this task arose in understanding how the VHDL processes work, and their differences between processes with or without sensitivity list. By making some experiments in VHDL code, I eventually learned their features and achieved the desired task.

3.3 Implementation

The top-level entity for this task, ***main2_four_digits***, has as input the 16 switches and the central button of the board. It will output an array of 7 bits representing the 7 segment to be illuminated, and the anode 4 bits vector for showing the respective digit.
It is an entity in which every group of 4 consecutive switches will be used by the user to represent each binary digit, and by the ***four_digit*** entity to be computed and decoded in the ***one_digit*** entity.

4 Task 1

4.1 Task Introduction

This task has the purpose to allow the user to use the rightmost 4 switches to input a binary coded digit that will be then displayed, and duplicated, on all of the four digits of the board.

4.2 Planning

The only one difficulty faced up during this task was to understand the pattern for illuminating the right segments on the board display. By using the **Figure 2**, which depicts the right sequence of the bits to be followed, this process was highly simplified.

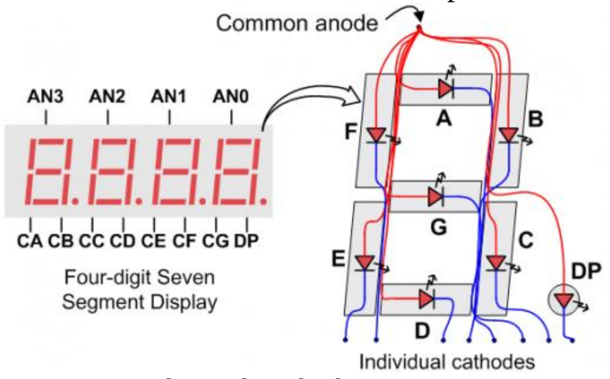


Figure 2 – *Anodes and Cathodes wiring representation*

4.3 Implementation

The top-level entity, which I was provided of, only takes as input a vector of 16 bits representing the 16 switches of the board; it will output then the 7-segment binary combination, represented by the rightmost four switches, the anodes linked to each digit represented by the leftmost four switches, and the point among each digit.
It is a simple combinational circuit which wires the binary coded digit jointly with the segment bit vector which will be used in the lower entity to decode the digit and convert it into a set of segment to be illuminated.
Inside the ***one_digit*** entity, I also created a particular constant, ***digit_E***, which allow to illuminate an E (as error) whenever the user attempts to encode a binary coded number greater than 9.

5 Conclusion

The purpose of this assignment was to gain familiarity with VHDL code and the differences between combinatorial and sequential logic circuits. The goal has been easily achieved thanks to the hierarchical structure which every task was structured in. By starting, in fact, by developing easier circuits I have been able to make use of the knowledges acquired task by task for implementing harder structures.

6 VHDL Code

CODE REMOVED