Lab Assignment - 6

(Egg Timer)

CS303 – Logic & Digital System Design

SPRING 2021/2022

Submission Report

Rebah Özkoç – 29207

Furkan Kerim Yıldırım – 28138

Problem Description

In this lab assignment, we are expected to design an egg timer circuit that will utilize and extend the previously designed 2-digit BCD down counter to incorporate minutes. In this circuit, when the button labeled “preset” is pressed, the controller is expected to load the preset value to the down counter, which is determined by the switches of the FPGA board. These switches must be grouped into four 4-bit blocks to enter the initial timer setting as BCD digits up to 99:59. The countdown should start immediately when "start" is pressed and stop at 00:00. The controller which conducts all these events will make the timer proceed in 1-second steps. This counter will be started by “start” and should signal the controller by the signal “endd” to indicate that a certain number of clock ticks have been counted. We designed this egg timer with a controller and a data path as an algorithmic state machine.

DownCounter.v

Down counter takes some inputs to perform the process. These inputs are a one-bit asynchronous "rst" input that controls whether the reset is triggered, a one-bit "load" input that controls whether the desired time value is loaded, a one-bit "count" input from the Counter that controls if one second is passed, a one-bit "c1khz" input that clocks with 1 kHz frequency, and the 2-bits "pm10", "pm1", "ps10" and "ps1" inputs that specify the desired value when the "load" input is triggered. The down counter returns the newly calculated time as two-bit outputs of "m10", "m1", "s10", and "s1", and returns a one-bit "zero" output indicating whether the result reaches zero. Also, the down counter creates the one-bit "zero_r" register and the one-bit registers "min10", "min1", "sec10" and "sec1" to calculate intermediate values and assign them to outputs.

The zero_r register is used to set the zero output to 1 when itself is 1, otherwise, return the zero output to 0. The remaining min10, min1, sec10, and sec1 registers are used to store values to m10, m1, s10, and s1 outputs.

The process of calculating the zero_r, min10, m1, sec10, and sec1 registers is done in a loop. This loop is executed every time the count and load inputs are posedge and the rst input is negedge. When it enters the loop, the min10, min1, sec10, and sec1 registers are set to 0 and zero_r register is set to 1 if the first rst is equal to 0. Otherwise, the load input is checked. If the load input value is equal to 1, pm10, pm1, ps10, and ps1 inputs and registers min10, min1, sec10, and sec1 are set. While setting these registers, pm10, pm1, ps10, and ps1 inputs are checked. If input pm10 is greater than 9, register min10 is set to 9, otherwise, the value of the pm10 input is directly assigned to the min10 register. If input pm1 is greater than 9, register min1 is set to 9, otherwise, the value of the pm1 input is directly assigned to the min1 register. If input ps10 is greater than 5, register sec10 is set to 5, otherwise, the value of the ps10 input is directly assigned to the sec10 register. If input ps1 is greater than 9, register sec1 is set to 9, otherwise, the value of the ps1 input is directly assigned to the sec10 register. Finally, the zero_r register is assigned to 0 for indicating to the result is not equal to 0. If the value of the load input is not equal to 1, that is, if the down counter is not in the load state, the min10, min1, sec10, and sec1 values are reduced accordingly themselves if the count input is equal to 1. If register sec1 is greater than 0, the value of register sec1 is decreased by one, and 0 is assigned to the zero_r register, otherwise, the sec10 register is checked. If register sec10 is greater than 0, the value of register sec10 is decreased by one, sec1 register is set to 9, and 0 is assigned to zero_r register, otherwise, the min1 register is checked. If register min1 is greater than 0, the value of register min1 is decreased by one, sec10 register is set to 5, sec1 register is set to 9, and 0 is assigned to zero_r register, otherwise, the min10 register is checked. If register min10 is greater than 0, the value of register min10 is decreased by one, min1 register is set to 9, sec10 register is set to 5, sec1 register is set to 9, and 0 is assigned to zero_r register, otherwise, 1 is assigned to zero_r register.

The values of the newly formed registers are assigned to the outputs according to the process described above.

Controller.v

Firstly, the sequential block of our program is sensitive to the positive edge of the clock and the negative edge of the “rst” input. If one of these is changed our sequential block run again.

For storing the program's current behavior, we used a register with 2 bits, namely “state”. This state can take 4 values, but we used only three of them. According to the value of the state, we changed the output appropriately.

The three states of our program were like that:

zeroState: When in this state the counter stops running and shows the loaded value in the screen if any.

oneState: When in this state the “start” output signal becomes true, and the counter module starts to run. After this state, the controller moves to “twoState”.

twoState: This state is used to wait until the “endd” signal comes from the counter module. When in this state the controller loads false to “start” output. After that, it waits for the ending of one second. At the end of a second, it moves back to the oneState.

In the sequential block of the controller, we also check the input values of the controller. If the “rst” input is low asserted, then the controller moves back to the zeroState and counter stops. If the “beginn” is high asserted and the zero is low asserted, then the controller starts to count by going the oneState. If the reset input is high asserted, then the controller moves to zeroState and counting stops.

The computational block of the counter is responsible for storing the count and start outputs in a temporary register. It also asserts the countdown signal if the counter module is ended and the state is not zeroState. (We check zeroState to stop the counter immediately after the rst input is low asserted).

Counter.v

Counter implementation is given by the professor to us, and it does not need any changes. This implementation is basically a clock divider. Counter gets a 1000 Hz clock and runs for a second and makes “endd” signal true until a new “start” input is asserted.

Eggtimer.dig

This digital circuit mimics the top-level module of our program. It contains the 4 seven segment displays, the downcounter and the controller module, and the necessary input values. It is responsible for combining all of the modules, inputs, and outputs.