ELEC 3500

Lab 9 – Egg Timer Project

Tawseef Patel 101145333

Individual Report

(Group Member: Saad Babur)

Dec 9, 2022

## Background and Design Goals (GA 4.1, 4.7)

When cooking/boiling an egg there are many different levels of cooked that one may want as in the case of steaks. With eggs the consistencies may range from runny to hard and the differences between these two far ends of the spectrum occurs within a matter of minutes depending on the cooking temperature. To ensure the accurate timing for a consistent cook level, a timer can be used. One such timer which was designed in this lab for this exact reason. The egg timer which has a precision of one second allows the user to set a timer to cook an egg for up to an hour. The design specifications for what was required when designing this egg timer using Verilog are mentioned in section 2.

## Design Specifications (GA 4.2, 4.7)

To provide a suitable solution for creating the perfect egg, the egg-timer was created and the requirements which were taken into consideration are listed below:

* The timer must indicate if the timer is enabled
* The timer must indicate if the timer is counting down
* The precision of the timer must be in the format mm:ss with a max time of 59:59
* To enhance user experience (innovation) the timer must be able to increment and decrement (0 to 59 if the user wants to press the buttons fewer times)
* The minutes and seconds must be able to increment individually
* The timer must have a reset button which allows for the timer to go back to 0
* The timer must have an enable switch and cooktime button to allow for user to begin inputting the desired time
* The timer must have a start timer button
* The timer must stop if the enable switch is disabled
* The timer must be accessible and be easy to use

These specifications allow for a seamless user experience when looking to use a timer to create the perfect egg.

## Design Implementation (GA 4.4, 4.5)

The design was implemented using Vivado which uses Verilog on the NEXYS 4 DDR development FPGA board. The board consists of multiple toggle switches, buttons, LEDs, I/O ports and pins, 2 quad seven-segment displays etc. The schematic of the elaborated design in shown in Figure 1. The design is driven using an internal 100MHz system clock which is divided down to 5MHz using the in-built IP Catalogs Clocking Wizard. This signal is then further divided down into multiple different clock signals for different functionalities. The 5000Hz clock is used on the quad seven-segment displays to ensure a fast enough refresh rate that the human eye cannot detect. While the 1Hz signal is used to act as a 1s timer to decrement the time at the proper frequency. A third clock signal of 100Hz is used for the debouncer which detects changes for the incrementing buttons (minutes/seconds). The mux for the quad seven-segment displays selects between the count time, and cooktime inputs. This helps store the values in the count time or cook time output registers so that the correct time is displayed. The bcdto7seg decoders allow for output binary values from the mux to be displayed on the seven-segment displays.

Diagram

Description automatically generated

Figure Elaborated Schematic of Master Control (top module)

### Detailed Block Descriptions

Diagram

Description automatically generated

Figure Block Diagram of the Egg Timer circuit

Master Control

The mater control module serves as the top-level module for the project. All the submodule instances are created in the master control and ties all the inputs and outputs together to feed into the submodules. There is some logic performed in the master control which ultimately could be in its own submodule (the quad seven-segment display decoder/initializer). Further, the master control has inputs: clk, enable (SW0), reset (SW15), set\_time (BTNL), minutes (BTNU), seconds (BTND), start\_time (BTNR), and the innovation up\_down (SW2). The outputs of this module are: count\_enabled (LED0), cathodes [7:0], anodes [6:0], eggtimer\_enabled (LED1), min (LED15), sec (LED14). The clocking wizard was used to divide the 100MHz system clock to 5MHz to ensure use of only 1 clock domain. The master controller also contains the MUX logic which decides if the count or cooktime needs to be displayed on the seven-segment display.

Clocks

The onboard 100MHz system clock which is routed using the .xdc constraint file is the main clock domain that is used and further divided down to 5MHz, 5000Hz, 100Hz, and 1Hz. This is because the system clock is too fast for the purposes of the eggtimer and cannot be used easily without lots of logical changes. This would reduce reliability which is why the clockdivider modules exist. The built-in clock divider uses Phase Locked Loop (PLL) to make it usable. The clockdivider modules have a threshold within it to serve as a counter-driven clock divider by having the outputs stay high for half of the desired period (50% duty cycle). The formula to get the divisor for clockdividers is .

Diagram

Description automatically generated

Figure Clock Divider Elaborated Schematic

Debouncer

The debouncer as shown in the asynchronous lecture slides follow a simple standard. The input to the debouncer is the set\_time button which is passed through 2 D-flipflops to synchronize the value to the 5MHz clock and then the outputs from both flipflops are passed through an XNOR gate to check for equality and then the output from the second D-flipflop and XNOR are passed into and AND gate to get the output. The debouncer makes use of a slower 100Hz clock because which is just fast enough to register user inputs but slow enough to not take each input from the user and multiple increments for counttimer. Running the debouncer at a higher frequency may create glitches in the circuit while the slower clocks would not let all inputs to be registered.

Diagram

Description automatically generated

Figure Debouncer Elaborated Schematic Design

Pulse Generator

The pulse generator acts as a synchronizer combined with some extra logic to generate a pulse of exactly one clock cycle. This is driven with the 5MHz clock to ensure the signals are properly synchronized. This is done by passing the signal through two D-flipflops and then each flipflop output is passed through an AND gate. This creates a pulse for every clock cycle, however, a millisecond pulse from the debouncer would last many pulses of the 5MHz signal which then can feed into count time and cook time modules to simplify the efficiency.

Diagram, schematic

Description automatically generated

Figure Pulse Generator Elaborated Schematic

Count Time

The count time module has 4 counters. From the LSB representing seconds to the MSB representing minutes. The LSB counter goes from 9 to 0 and loops back up to 9. When the loop around is happening, 1 is subtracted from the MSB which counts from 5 to 0 and back to 5. The seconds bits are tied to minutes, so after 60 seconds, the minutes decrement. The counter is complete when all values equal to 0 and the output is held at while the FPGA awaits for the new count values to be loaded in. When a new value is loaded in, the counter resets and starts counting and the outputs are once again selected from the mux and displayed on the segment display. The full schematic for count time can be found in the appendix.

Cook Time

The inputs to the cooktime module are clk, enable, reset, increment\_second, increment\_minute, and up\_down. The outputs are seconds\_out1, minutes\_out1, seconds\_out2, minutes\_out2, which are 4-bit registers representing binary numbers from 0 to 9. The reset for the cook time allows the user to reset the time that has been set at any given time on the posedge of the clk. To decrement the time (innovation) allows the user to decrement the displayed time if the increment\_minute/second are both enabled, along with the main enable signal and the up\_down enable signal. The logic for the increment timer works the exact same way but without the up\_down enable signal needing to be 1.

Timer On

The timer on module simply takes the start and count done signal from the rets of the circuit and displays a flashing LED to signify if the timer is counting down or stopped. The LED blinks at a rate of 1Hz only while the egg timer is counting down.

Diagram, schematic

Description automatically generated

Figure Timer on Elaborated Schematic

Display Logic and MUX

The display logic is made up of 3 portions, the MUX, BCD Decoder, and anode driver. The MUX selects between the outputs of count\_time and cook\_time with two 4-bit wire inputs and gives an output of one 4-bit bus (2 to 1 mux). The resultant signal from the MUX feeds into the BCD decoder and converts each 4-bit value from the bus into a Binary Coded Decimal of 7 bits corresponding with the cathodes of the seven-segment display. For example, 4’b0010 representing 1’d2 requires the cathodes to be 7’b0100100. The anodes can only be enabled once at a time which is why the 500Hz clock display is used to drive the anodes to make it seamless for the human experience. The anodes are enabled and take a new value with an active low signal.

Diagram

Description automatically generated

Figure Display Logic and MUX display logic Elaborated Schematic

## Evaluation (GA 4.7)

The design was evaluated at multiple points throughout the process due to changing requirements after building each block. Since each module was programmed separately, there was a need ensure compatibility between all of them and thus this caused us to re-evaluate how each module interacted with one another and what inputs would be fed in and what outputs are required. The final design which was used allowed for maximum flexibility amongst communicating modules.

### Testbenches

To test the functionality of each module, a testbenches were individually developed for each module and tested to make troubleshooting easier. This was because, during the evaluation stage after making changes to the modules to make communication easier, there were errors within the code which caused improper functionality, however, with the use of the testbench simulations, allowed us to find common places where errors were occurring. While test cases for each possibility is not feasible, there were direct tests which were completed and then randomized to get the maximum coverage with minimal work. The modules which were proven to be successful in previous labs are not shown below.

Debouncer

A screenshot of a computer

Description automatically generated with medium confidence

Figure Debounce Module Test Bench

The debouncer is one of the most important functionalities of the eggtimer circuit, however, using software debouncers, reduces the need for a hardware module. The debouncer module was thoroughly tested to ensure the functionality would work on the board.

At 15ns, its noticeable that a few glitches occurred when it wasn’t supposed to, this was because the pulse signals were not long enough and require a hold of 2 clock cycles to propagate to the output. This is why it is important to have a clock frequency which allows for multiple clocks within a second so that the user would not notice if it didn’t work at the first clock cycle as it would be difficult to do things in under a second due to human limitations.

Pulse Generator

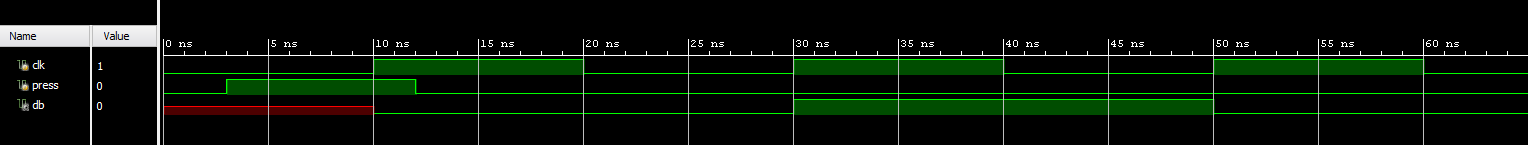


Figure Pulse Generator Testbench

The pulse generator module synchronizes the inputs to generate a 1 clock cycle pulse. This is shown in the above figure.

At 10ns the input is latched and propagates to 1 clock cycle until 30ns. The resultant pulse is exactly one clock cycle in length.

Count Time

Graphical user interface

Description automatically generated

Figure Count Time Testbench

The count time module is one of the core modules of this circuit and required extensive testing for all possible inputs to ensure functionality. Initially, the reset is checked to confirm functionality and makes all inputs go low. Following this, the count done signal is expected to go high on the following clock edge because all the inputs are 0. Once a number is loaded into the timer, it is expected that the output is decreasing if enable is high.

The reset starts off high and the count\_done is 1 until the number is loaded into the register with set. The count does not start until the 20ns point when both enable and start are pulled high until the output reads 0 and then holds that value.

Cook Time

A screenshot of a computer

Description automatically generated with medium confidence

Figure Cook Time Test Bench

The second key module for this program is to store and set the cooktime. When enable is low, the circuit should not activate. This ensures the user is only setting the time when they explicitly specify it with the press and hold of cooktime button. When the enable is high, the registers are set with the value by having pulses from the pulse generator module send a virtual enable increment signal. These values are stored and outputted.

The minutes and seconds both increment with each press of the button and reset sends everything back to 0. There is also no response when the enable signal is low.

## Alternate Solutions (GA 4.6)

There are always ways to improve the design to enhance user experience. After considering how to improve, there were a few changes that were made: The first change that was made was regarding the start/stop logic for the timer. It is inconvenient for the user to use the “set” button to stop the timer because it also resets the time to the original set time. This allows the user to pause the time and restart from where it was stopped as most other real world timers work. The second change was implementing the innovation which was an up\_down toggle switch for the user. This allows the user to decrement the time if they go past their desired time with the fewest amount of button clicks. These changes enhance user friendliness and functionality of the timer.

Start Toggle

The start toggle function is an improvement to the start pulse module which allows the start signal to toggle rather than hold.

## Innovation

Up\_down toggle

The up/down toggle is an extra signal that is fed into count\_time that allows subtraction to the count registers. When this toggle switch is high, the register decrements the value.

## Conclusion

Overall, the egg timer was successful implemented and tested against the outlined requirements that are mentioned in an earlier section. The results ended up meeting and surpassing the expectations as the functionality was improved upon to enhance the user experience by offering multiple solutions through accessibility and inclusivity. This solution was the final solution and it allowed me to learn how to properly implement multi module digital circuits while using test benches and actual hardware to compare results. With these lessons, the further improvements that can be made is to better understand the goals that were laid out in the lab manual to understand the project effectively.

Diagram, schematic

Description automatically generated

Figure Count Time Elaborated Schematic

Master Control Code

Text

Description automatically generated

Text

Description automatically generated

Master Control Code CNTD

Text

Description automatically generated

Cook Time Code

Text

Description automatically generated

Text

Description automatically generated

Count Time Code

Text

Description automatically generated

Text

Description automatically generated

Clock Divider Code

Text

Description automatically generated

Debounce Code

Text

Description automatically generated

Mux Control

Text

Description automatically generated

Start FF / Pulse Gen Code Code

Text

Description automatically generated

Timer On Code

A screenshot of a computer

Description automatically generated with medium confidence

BCD to 7 Segment Display

Text

Description automatically generated