Alarm Clock with Buzzer

List of Authors (Brendan Alkevicius, Trey Plichta, Samuel Bejko, Andrew McGhee)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: balkevicius@oakland.edu, treyplichta@oakland.edu, samuelbejko@oakland.edu, amcghee@oakland.edu

***Abstract*—ASM Implementation of an alarm clock with buzzer. LUT utilization for time display. FSM implementation is necessary for proper function of the system. Current state of project design allows for minutes/seconds counter and alarm.**

I. Introduction

This project's aim is to create an alarm clock using 8 separate 7 segment displays on an FPGA. The project heavily relies on FSMs. The project also relies on knowledge of full adders, decoders, encoders, priority encoders, multiplexores, d flip flops, hex-to-7 segment decoders, LUTs, and logic gates. The project includes a self-drive piezo buzzer which will require the use of a pinout pin on the FPGA board, which was not covered in class. A piezo buzzer will be used to indicate when the alarm goes off notifying the user. A single button will turn off this buzzer or it will automatically deactivate after 5 minutes. A single button is used in addition to four switches to set the time of the alarm and/or clock. The alarm clock will be based around several FSMs. These are the Button FSM, the FSM controlling the numbers in the 7-Segment Display, the FSM controlling the buzzer output, and the FSM controlling the stop buzzer button.

II. Methodology

A. Button Press

The algorithmic state machine diagram for the mechanism of the button press that increments the clock or the alarm time is as follows:

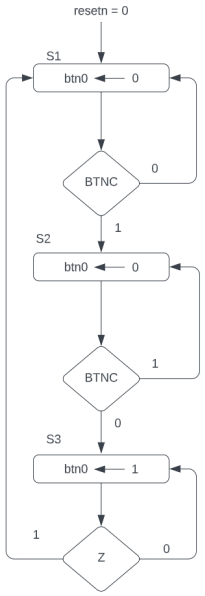


Figure 1. Button Finite State Machine.

The FSM is divided into three states: S1, S2, S3.
Once resetn is achieved the value of '0', the button's state will be S1 until it is pressed. The value of the BTNC is then checked. If the value of BTNC is high, the state increases to state 2. If the value is '0' the process will return to state 1. The second state acts as a debouncer for the button press. Since btno is only desired to be high for one clock cycle, it does not make sense to have btno equal to '1' for the entirety of S2. As mentioned, S2 again sets btno <= '0' but instead if BTNC is '0' the process will advance to state 3, otherwise it will stay in state 2, until the button is released. In S3, btno will now have a value '1' since at this stage the button will have been pressed and subsequently released. At state 3, z (the output of the onboard clock) is also checked to either keep the process at state 3 or proceed back to state 1. The FSM outputs btno to be '1' for exactly one rising clock edge of the onboard clock. This all allows the rest of the circuit to function properly and only count once when BTNC is pressed.

B. Demultiplexer with Enable

btno acts as the 1-bit output of the FSM "BTN". Based on the selector value, it picks the value of the selected bit and assigns it to a specific output, as shown below.

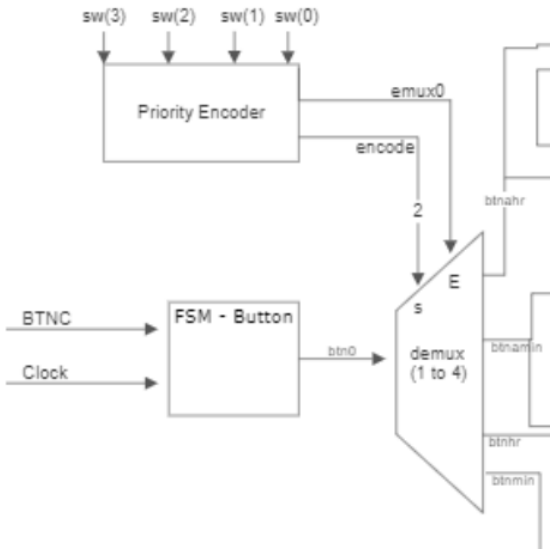


Figure 2. The Button Demultiplexor with Enable is displayed.

The enable for the demultiplexor represents the single bit *z* output of the priority encoder. The 2-bit output 'encode' acts as the selector for the demultiplexer. The priority encoder utilizes the switch inputs as a selector for the clock/alarm by being encoded and sent to the aforementioned demultiplexer. The switches in the priority encoder are utilized to switch which displayed value is to be modified. The FPGA board's eight 7-segment displays are split up into 4 groups: alarm time hours, alarm time minutes, clock time hours, and clock time minutes. Each switch corresponds to a different group mentioned in prior on the on-board 7-segment displays as seen in the de-multiplexor with the priority going to the highest number switch that is high. The button's purpose is to simply increase that certain value every time it is clicked.

C. Clock/Alarm Mechanism

Figure 3 shows the main mechanism for the clock time and alarm time. The clock & alarm consists of four counters of four different types; modulo-2 for the hour ten's position, modulo-12 for the hour one's position, modulo-5 for the minute ten's position, and a BCD for the minute one's position . The one's hour position had a specific issue. In that specific spot the hour ten's position has to count from 1 to 9 and then 0-2 before restarting. Additionally, this part of the circuit had to somehow send a signal to the modulo-2 counter to restart on the count when the hour one's position reached 9 and when it reached 2. In order to achieve this effect, an LUT and a 4-bit adder with constant inputs 0001 as x and 0 as cin were used with a counter. The LUT's first bit (MSB) was used as the messenger bit to the modulo-2 counter above it. The 4 other bits were used as the output for the hour one's position and were sent to the next part of the circuit; the 7-segment serializer.

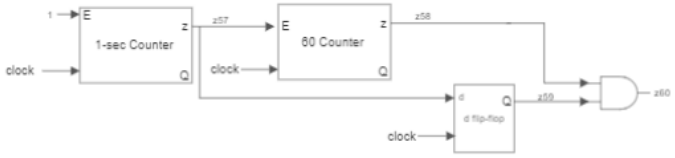
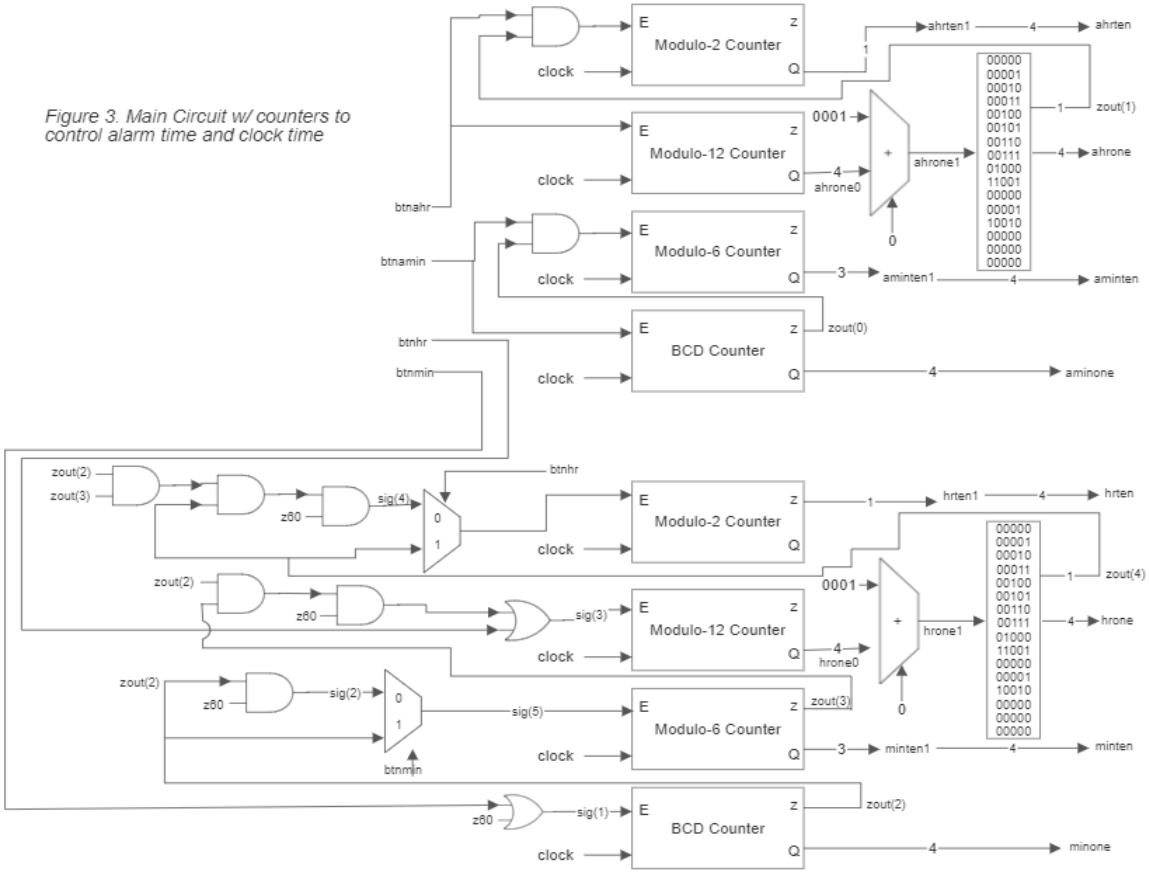


Figure 4. Cascaded 60 second counters

As it relates to the clock time, all of the counters clocks run off of the FPGA onboard clock. However, the enables of the counters are where the magic happens. Each enable is driven by multiple sources. The enable signals are controlled by a button press(release) and a counter. There is a series of cascaded counters that has an output (z60) every 60 seconds, controlled by the FPGA board's clock, shown in *Figure 4*. The button press signals are btn_hr and btn_min in figure 2. There is some added circuitry in order to get the functionality of a clock. Beginning with the BCD counter, an or gate is added that activates the counter either when the user tells it to or when a 60-second counter reaches its maximum value. For the modulo-5 counter, the 60-second signal (z60) is anded with the z output of the BCD counter. This allows the modulo-5 counter to increment when both z60 is high and the BCD counter hits its maximum value. The multiplexor is added in with a select line that is controlled by a button press(release). This allows the modulo-5 counter to only look at the BCD counters z-output as its enable for one clock cycle after a button press with the switches set to control the clock time's minutes. The modulo-12 and modulo-2 counters are grouped together in a

Figure 3. Main Circuit w/ counters to control alarm time and clock time



similar fashion with similar components. However, more and gates are used to look at the z outputs of the other counters.

As it relates to the alarm time, the clocks of the counters are once again run off the FPGA onboard clock. However, the enables of the counters are controlled purely by the button mechanism described prior. The button press signals for the alarm are btna_hr and btna_min in figure 3. Since the modulo-2 and the modulo-12 counter are grouped together, the button press signal is sent through an and gate with the other input being a one bit output from the LUT to tell the modulo-2 counter to activate and increment on the next clock tick. Since the modulo-5 and the BCD counter are grouped together, the button press signal is sent through an and gate with the other input being the z output from BCD counter to tell the modulo-5 counter to activate and increment on the next clock tick.

D. Full 7 Segment Display

In figure 5, the FSM and attached components are used to show all of the signals on separate 7-segment displays to the human eye simultaneously by cycling through every 1ms(0.5ms may be necessary). The state diagram for the aforementioned FSM is shown in figure 6. The FSM increases its state every time E is 1, which as the diagram shows, will occur every 0.001s (1ms). Hence, ‘s’ will increase every 1ms, thus switching what data value to fetch from the multiplexor each millisecond. The ‘s’ selector also selects which 7-segment display to turn on at a given moment since only one display can be utilized at a time because of hardware limitations. Hence why the 1ms delay is present to let the board switch displays constantly so it is only utilizing one display at a time while simultaneously seeming as though the image is still to the human eye thanks to the low delay.

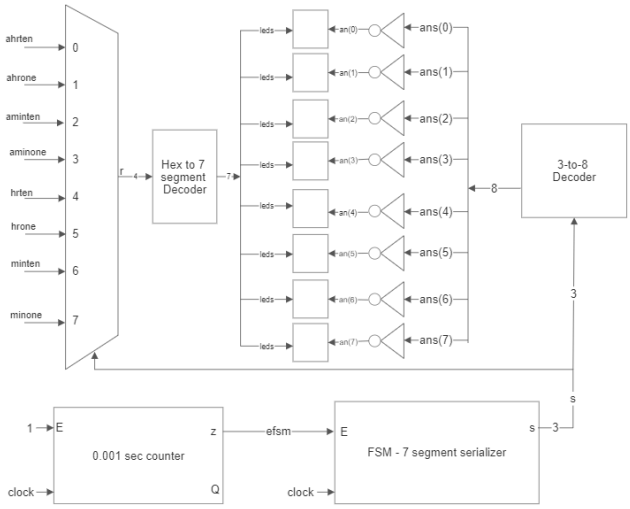


Figure 5. Clock/Alarm mechanism outputs through decoder to 7-Segment display. MUX, Counter, FSM, and 3-8 Decoder.

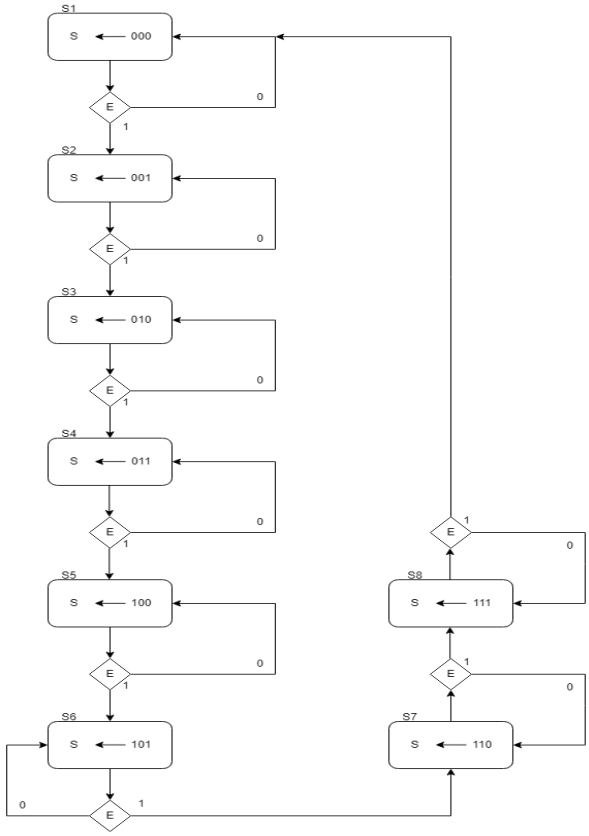


Figure 6. State diagram for FSM.

E. Buzzer Circuit

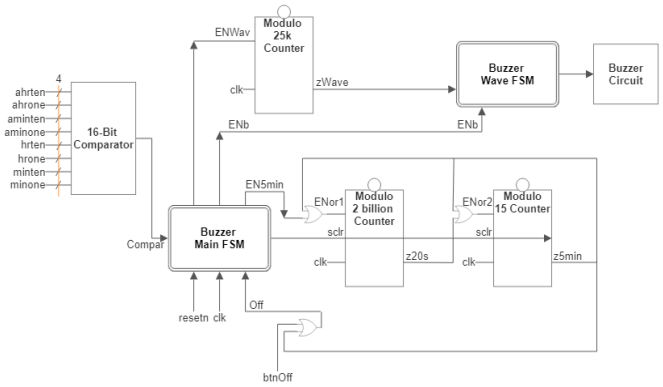


Figure 7. Detailed buzzer circuit is shown.

For the buzzer circuit, two FSMs were used to control the buzzer functionality. The Buzzer Main FSM was utilized to determine whether the buzzer should go off based on the inputs. These inputs are a comparator, btnOff, and z5min as shown in the following figure.

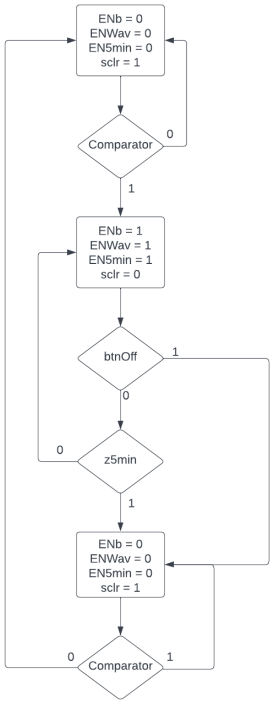


Figure 8. Buzzer Main FSM

The output EN5min, that will make the buzzer go off for up to 5 minutes, then acts as the enable for the 2-billion-modulo counter when compared in an or gate with z5min. The two cascading counters are present to bypass the hardware limitations values for a singular counter, allowing the counter to reach far higher values by chaining them in such a way. For context, the modulo 2-billion counter, counts up to 20 seconds, and the modulo-15 counter allows for the repetition of that 20 second period for a total of 15 instances for a total of 300 seconds, or 5 minutes. The z5min signal is connected in an or gate to the input to ensure that the sclr signal from the buzzer control FSM resets the counters to 0 before the next cycle.

Another output of the Buzzer Main FSM is ENWav, which is responsible for being the enable of a modulo-25k counter that sends out a zWave signal as a signal for the Buzzer Wave FSM pictured below. The modulo-25k corresponds to a 4kHz, 10 ns pulse which equates to a 2kHz full square wave which generates the audible buzzer tone.

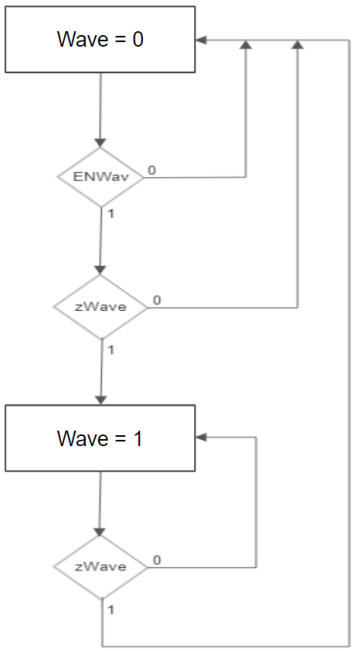


Figure 9. Buzzer Wave FSM

This simple FSM relies on both ENWav, the enable for the modulo-25k counter, as well as zWave, the output of said counter, to have a value of '1' in order to generate the 2kHz frequency waveform in the second state. If zWave is then again triggered, the wave generation will stop.

III. Experimental Setup

To attempt to implement this alarm clock an Artix-7 FPGA was used. The software used to implement the design was Vivado VHDL by Xilinx. The configuration uses various components including everything mentioned above. The expected results for the first test was a working clock with a 7-segment display being able to count at a base of 1 minute increments. In addition to that, switches and button presses in order to increment the minutes and hours section of both the alarm and the clock. For the implementation of the buzzer, an HF-12095, was utilized and connected to the pinout of the Artix-7 board.

IV. Results

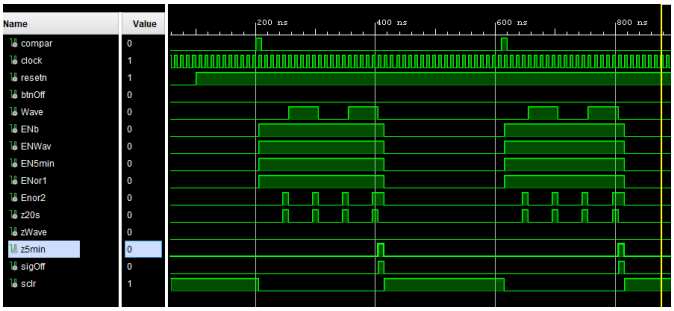


Figure 10. Buzzer Circuit waveform and internal signals display the functioning system.

The various FSMs were of great importance in successfully implementing the various counters needed for this project especially when dealing with clocks. All the FSMs in the main clock circuit, time adjustment button circuit and datapath circuitries functioned as expected and no major flaws surfaced.

Figure 8 shows the simulation of the comparator and buzzer circuit. The values of the counter to turn off the alarm were scaled to be active in this shorter window, as well as the actual waveform used for the buzzer. However the simulation shown accurately portrays the use of "z5min" to send a signal to the control FSM to switch to the off state when it reaches a logic high.

Some issues encountered during the testing process were noted and adequately addressed. One such issue was the problematic size of one of the counters. The 60-second counter reached bit numbers that were higher than what the FPGA hardware could handle (31 bits, 2^{31}), therefore 2 counters had to be cascaded in addition to a d flip flop and an and gate to create a counter that lasted for the correct duration.

Another issue that was encountered was due to the comparator comparing only the minute and hour values. This meant that the comparator value would stay high for a minute, or until the two values were no longer the same, preventing the alarm from being shut off within the first minute of activation. This was resolved using an added state to an FSM to prevent the alarm from sounding once the turn off button was pressed during the same minute the alarm started. This could have been fixed by incrementing an internal counter in the seconds place and using that to make the two compared values 24 bit rather than the 16 bit integer that's currently being used to set off the alarm.

The project required the use of LUTs in order to successfully implement a properly functioning ones digit for the hour in both the alarm and clock. Due to the nature of the ones place, it would need to count differently than a typical modulo counter, so a 4-5 LUT was used in order to properly increment the value.

Video Demonstration

Conclusions

This project introduced us to a deeper understanding of VHDL code and the design and implementation of digital systems. We worked with and expanded our knowledge on the pinout of the Artix 7 100T, which was not discussed in class. Our project ended in a completely functional state, but further improvements could be made. A snooze functionality was not implemented, but if more time was available, it would have been next on our list. This would add 5 more minutes to the set alarm time while also triggering the stop alarm state in the FSM. Additionally, there are some small simplifications or improvements that could be made. Our comparator is more complicated than it needed to be. Rather than folding down the output of the xnor gate, the output could have been put into a 16 input and gate, which would have the same functionality while only using one gate. Through this project, the group gained knowledge and hands-on experience with the design of a digital system that dove deeper than what the labs in the course provided. Knowledge from the labs built an understanding of how systems work, and this was used to create this alarm clock. The group is happy with where the project ended, and look forward to advancing our knowledge in FPGAs and their applications.

References

[1] Reconfigurable Computing Research Laboratory. (n.d.). Retrieved November 15, 2022, from http://www.secs.oakland.edu/~llamocca/index.html