Name: Cody Morse

Date: 3/29/2021

Course: ECE 631

Assignment: Lab 7

Introduction:

Lab 7 was not particularly difficult in terms of code and theoretical. The hardware was basic as all was required was: multiple wires, Analog Discovery 2, esp32, Raspberry Pi 4, and an ultrasonic sensor (hc-sr04). At the beginning of lab, the trigger signal needed to be verified as it needs a minimum of 10us trigger pulse width and a 60ms duty cycle before the next pulse. This was verified using the analog discovery 2 utilizing the waveforms software. The testing section of this document will show the pulse width output used to verify that the ultrasonic sensor would function properly. Once verified, we could then proceed with the code to perform our designated operations.

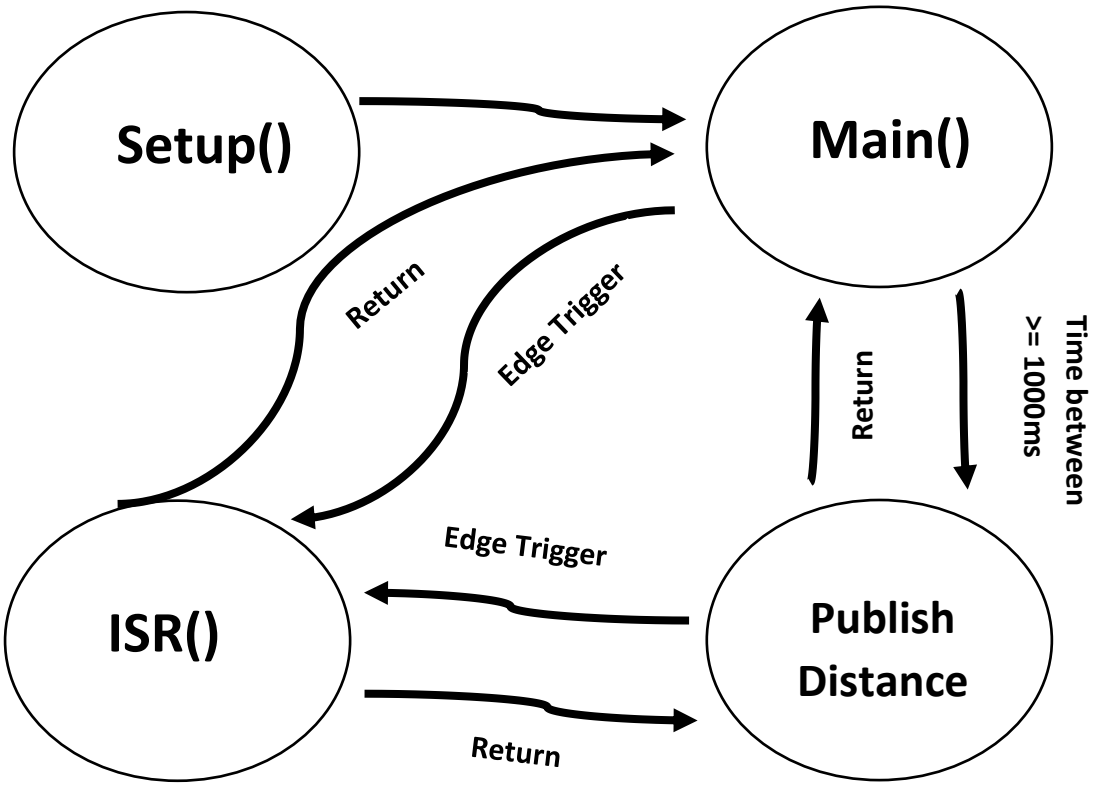
The hardware goal of this lab was to send a trigger signal to the ultrasonic sensor then transmit an echo signal back to the esp32. We verified that the esp32 was receiving the signal by using the analog discovery 2 and seeing if the ultrasonic sensor was adjusting the pulse width. Then the esp32 sends a compiled json to our Raspberry Pi 4 MQTT server.

The initial setup required to complete the lab requires code used in previous labs. We setup our Wi-Fi and MQTT server utilizing the PubSubClient and WifiClient libraries with the exact code from old labs. The Wi-Fi client was setup using our lab Wi-Fi username and password and our MQTT server was setup using our Raspberry Pi's static IP address.

Once connected to the MQTT server the overall goal was to first utilize the pulse width modulation specifications provided at the beginning of the lab to send a signal to the ultrasonic sensor, then once the echo rising edge signal is transmitted back to the esp32 an interrupt occurs and handles the pulse width calculation. The interrupt occurs each time the ultrasonic sensors pulse switches edges. For example, if the ultrasonic sensor pulse going high to low or low to high, then the interrupt service routine will be called. Utilizing the calculated the pulse width we can calculate the distance using the provided function in lecture. Our distance calculation is as follows: distance = ((width / 1000000) * 13503.9) / 2. This distance is then implemented via the moving average filter. The moving average filter is starts with an array with a specific number of elements based on the number of samples. Once the number of sample distances has been reached then the rolling average is computed. This reduces the effect of noise and gives a more precise distance measurement. Finally, once a distance is computed, then that distance is serialized and published to the Raspberry Pi's MQTT server (IP = 192.168.1.120).

Design Partitioning:

In the design for lab 7 there is four states created. Below is a diagram showing the full state diagram. Each state partition will be given reasoning and explanation below. The code was partitioned this way because it made the most sense as far as implementation purposes. Based upon the instructions for lab 7 only four states seemed to come to mind to fully implement the ultrasonic sensor and MQTT server publication.



Beginning with the setup state we setup the PWM output to be on channel 0, given an 8-bit resolution, and a 16hz frequency. This state happens each time the program begins on the esp32. This output is assigned to the trigger pin (pin 23). Additionally, an interrupt is attached to the echo signal pin (pin 22) and the trigger pin is given our specified duty cycle. As in previous labs, the MQTT server and Wi-Fi is setup. This concludes the setup state of our system.

Next, is our main state that begins once the setup state is complete. The main state consists of our moving average filter and a substate that publishes to the MQTT server. The main

state can be interrupted by the echo edge trigger from pin 23. This will go to the interrupt service routine and then return to the main state. The main state will proceed into the publish distance state each 1000ms, then once the publish distance state is complete, the current state will return to main state.

Additionally, we have the publish distance state which is a substate of the main state. The publish distance state is only entered each 1000ms or be interrupted by the edge trigger from the echo pin, then return to main state once completed. This state takes our rolling average distance and uploads it to the MQTT server.

Finally, the interrupt service routine is the most important state of our system. This state includes calculating our width and distance based upon the specific edge that is triggered. If this state is entered, then the state can return to main or publish distance based on where the code execution was interrupted from.

Design Issues/Limitations:

There were not many issues encountered when creating the code besides working with the interrupt, implementing the moving average filter correctly, and sending serialized json. These three issues were relatively easy to fix because the changes required to fix them were small.

First, the issue with the interrupt occurred because the interrupt trigger was not set to change, and the speed of light was used instead of speed of sound. The code was much longer with not having “CHANGE” as the interrupt trigger, so the code was implemented using “CHANGE” and reduced the lines of code yielding the same output as before. In addition, the speed of light was used in calculations instead of the speed of sound. Based on the lecture notes there was an equation for distance which included the “c” variable so in my head the 3x (10^8) m/s value seemed appropriate. In conclusion, the wrong value was chosen and changing the variable to the speed of sound resulted in correct output.

Second, the issue with the moving average filter occurred when implementing the code for MAF inside the 1 second delay time conditional. This issue was fixed very easily by simply moving the moving average filter outside of the conditional at the top of the main loop. In conclusion, the output was correct once implemented correctly on serial and MQTT server.

Finally, the issue that did not really pertain to overall functionality was the publishing of the json to the MQTT server. The issue was not being able to json.dumps( ) as we could in python which required research to implement in the Arduino IDE. To fix the issue a temporary dynamic json document was created then assigned two keys for distance and units along with their values inside of the main loop. Then, the json document was serialized and converted into a char array. With the char array the data can now be sent via esp32 to the Raspberry Pi's MQTT server. In conclusion, the MQTT received the messages in the proper format.

Testing:

There were not many functions of the system to test besides the following: proper pulse width and duty cycle before implementation, echo pulse differentiation, and MQTT server publishing.

First, testing for proper pulse width and duty cycle required us to utilize the analog discovery 2. As stated in the introduction part of this document the trigger pulse required and duty cycle required is 10us and 60ms, respectively. In the waveform's software, the time/division was spread decreased to see the approximate trigger pulse width, which in fact was greater than 10us given our standard PWM specifications. Then, the time/division was increased to see the duty cycle and was verified to be greater than 60ms. So, with our PWM and trigger signal the ultrasonic sensor will meet the required specifications. This is shown in the waveform plots on the following page in Figures 1 and 2 as indicated by the blue line.

Second, testing for echo pulse differentiation based on ultrasonic sensor distance was done on the analog discovery 2. As this was not required, seeing the echo pulse width change with distance from ultrasonic sensor was useful in verifying our sensor was working properly. In addition, testing this allowed me to understand the process more theoretically. Thus, the output was tested and verified based on distance from the ultrasonic sensor. This is shown in the waveform plots on the following page as indicated by the orange line.

Lastly, testing for MQTT server publishing we had to test if the moving average filter impacted the response time and if the server was receiving the messages properly. There were many numbers of samples we implemented that changed the behavior of MQTT server output. We tested the following number of samples: 5, 16, 100, 1000. Both 5 samples and 16 samples yielded instant arrival time each second when published to the MQTT server. As we only have a

16hz frequency both 100 samples and 1000 samples took longer than 1 second to arrive at its average value. This was expected as described by the lab specifications. Testing the 1000 samples was producing results that took too long to arrive at its average distance, so this was cut off to save lab time. Thus, having a sample count less than or equal to 16 will be optimal for producing an average distance within the 1 second interval between publishing to the MQTT server. The MQTT server publishing was tested utilizing our serialized json to char array which yielded the correct json output in the MQTT server. On the following page in Figures 3 and 4 the correct output is tested with 5 samples in MAF on both the MQTT server and serial, respectively.

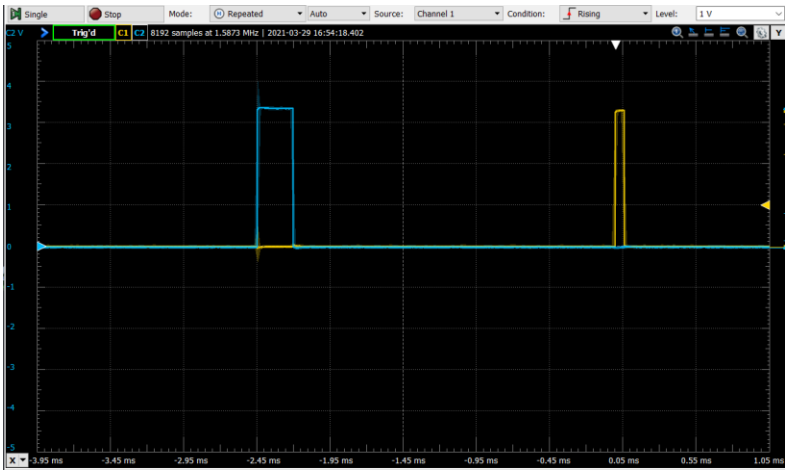


Figure 1: Waveform showing both Trigger Pulse and Echo Pulse

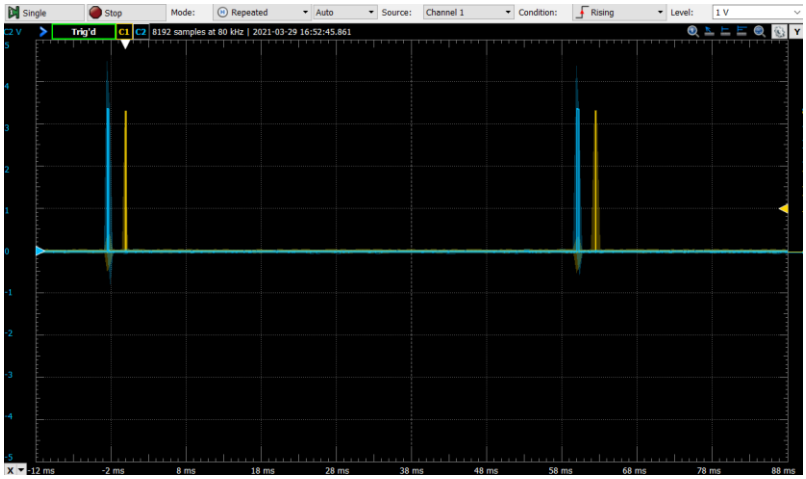


Figure 2: Waveform showing the duty cycle between each Trigger Pulse

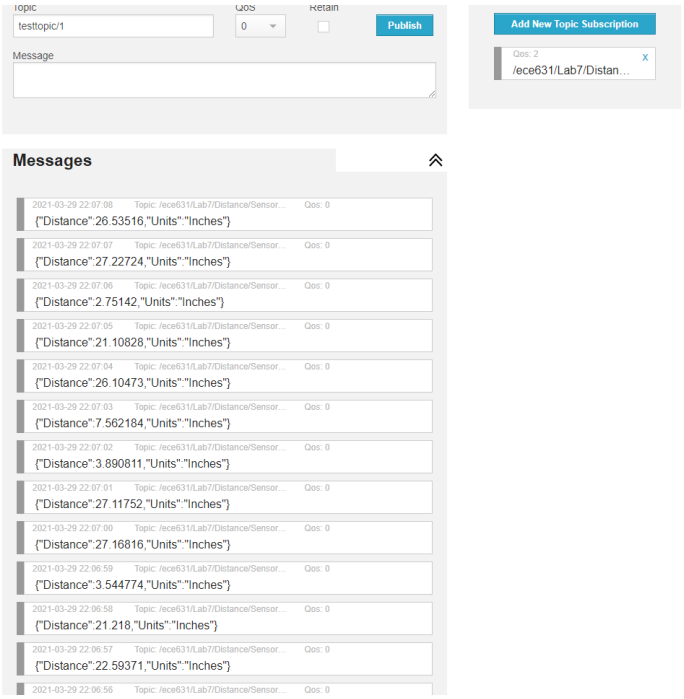


Figure 3: MQTT server output using a 5 sample MAF

```
.
WiFi connected
IP address:
192.168.1.43
Attempting MQTT connection...connected
ESP32 Lab 7
Distance: 2.77
Distance: 5.24
Distance: 7.09
Distance: 55.52
Distance: 13.95
Distance: 27.10
Distance: 10.41
Distance: 26.02
Distance: 3.56
Distance: 27.15
Distance: 5.39
Distance: 23.93
Distance: 27.02
Distance: 7.55
Distance: 7.07
Distance: 9.31
Distance: 7.56
Distance: 6.93
Distance: 7.40
Distance: 7.39
```

Figure 4: Serial output using a 5 sample MAF

Conclusion:

In conclusion, the following steps are taken to ensure proper output to the MQTT server (in order). In setup pin 23 is assigned as the trigger pin and pin 22 is assigned as the echo pin with an interrupt attached. Beginning loop execution, each time there is a low-high and high-low edge occurring in the echo signal the execution is interrupted and pulse width and distance is calculated. The distance is implemented into the MAF until the number of samples has been reached, then the average distance is computed. This distance is serialized and published to the MQTT server each second. This lab allowed me to utilize my critical thinking skills and knowledge gained from lectures.

If there was something, I could do to improve my build of the project I would remove the code I copied from lab 5 as it would increase overall code readability. If my future employer were reading my code this would be very important to account for. The reason why the code was left in was because it might be potentially useful in the next lab depending what lab 8 entails. In addition the serialization of the json document could be potentially done in a more efficient way because in my approach may not have been the most efficient.

Appendix:

Appendix A: GitLab/Github Repo Link

GitLab: https://gitlab.beocat.ksu.edu/s21.cody598/labs.git

GitHub: https://github.com/cody598/ECE631.git