CSC 460
Phase 1 Report
January 26th, 2015

Paul Hunter
Justin Guze

1.0 Introduction

In this report we detail the work undertaken to complete Phase 1 of the CSC 460 Design Project. The scope of the project involves creating a system capable of interfacing with an IRobot Roomba drive train to play a game of tag with other group submissions from the class.

In Phase 1 we were required to create a transmitter system in which an infrared, IR,emitter, positioned using a joystick could be used to transmit a data packet to an individual receiver; a communication protocol specification provided was to be implemented to allow the operator to send a single byte of data to the receiver. This report will detail the design of our solution system, including discussion of the mechanical/electrical hardware used, software, timing constraints, the joystick and servomotor driver implementation, and finally the IR transmission protocol implementation.

2.0 System Overview

In this phase of the project, our Arduino board is connected to three external devices, the joystick, servomotor, and IR emitter. The block diagram below shows a simplified view of the connections between each component. Power lines are omitted and data lines are shown with arrows indicating the flow of information. Figures 2 through 4, which follow below, provide a detailed wiring diagram for each peripheral interfaced with the Arduino system.

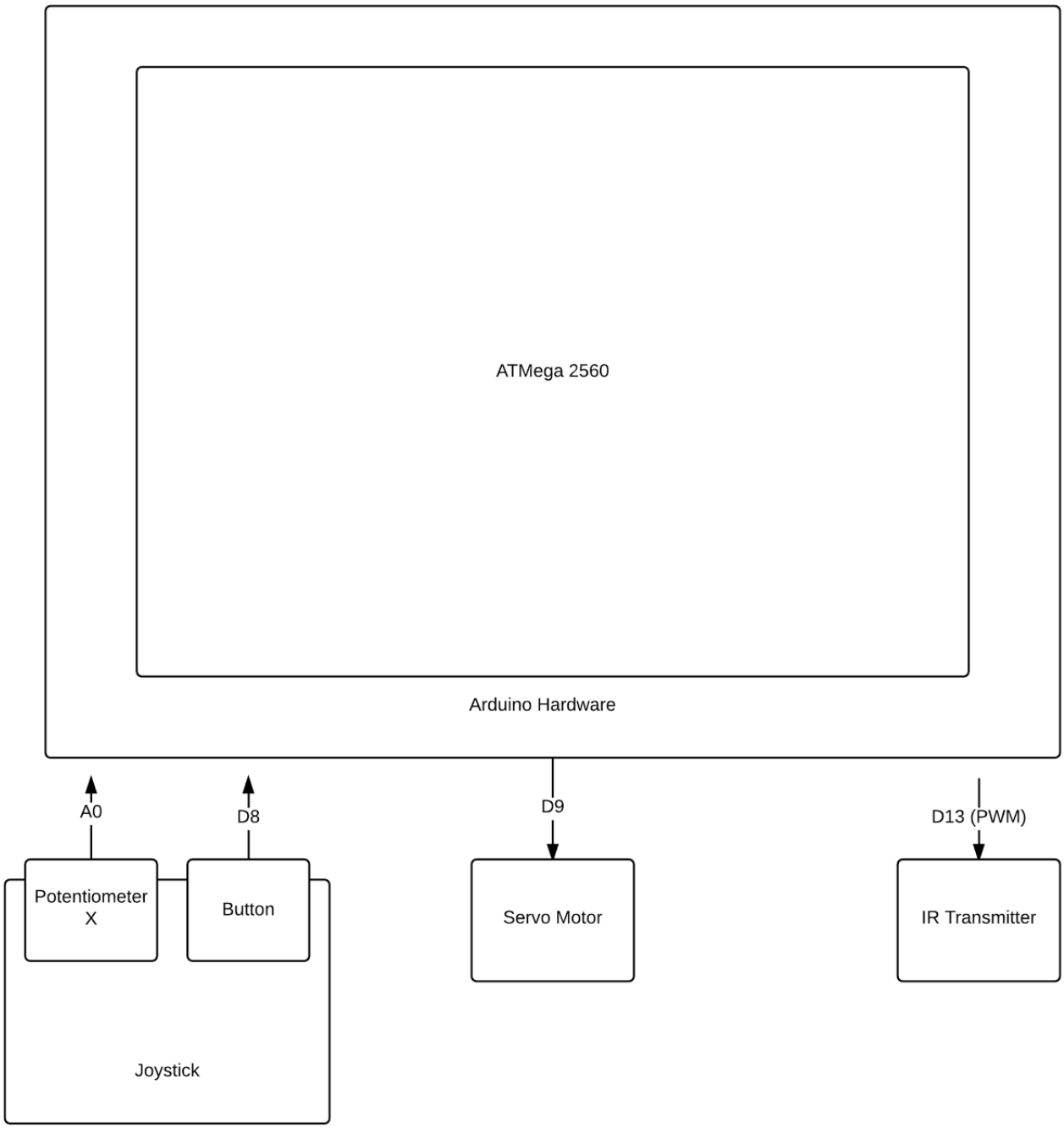

Figure 1 - High-level block diagram of system components

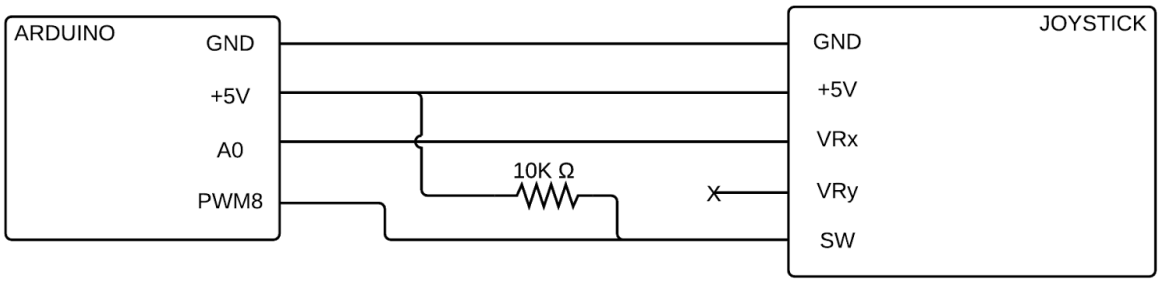

Figure 2 - Wiring diagram of Joystick

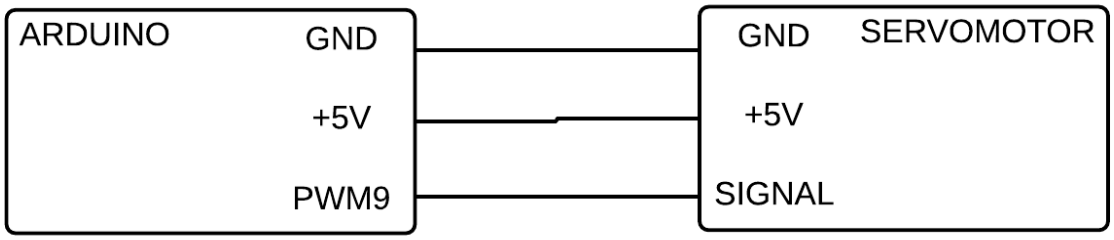

Figure 3 - Servomotor Wiring Diagram

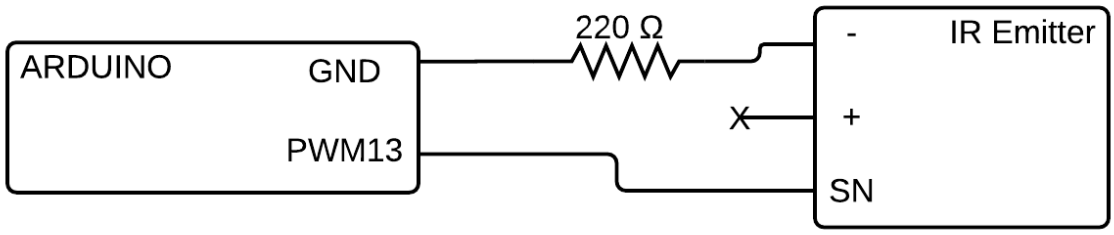

Figure 4 - IR Emitter wiring diagram

3.0 Joystick and Servo Motor

For the integration of the joystick, we chose to use only the X axis potentiometer as input for the positioning of the servomotor. We also utilized the included switch, present under the joystick, to trigger the transmission of a byte sized packet to a receiver. To ensure accurate sampling of the switch we added a 10K Ω pull-up resistor to the button circuit, shown in Figure 2, which was sampled on the PWM8 pin. The X axis of the 10K Ω potentiometer was sampled using the A0 analog input..

We initially integrated the servomotor making use of the existing Arduino servomotor. Unfortunately when we investigated the library we found that it utilized all three timers of the ATmega2560, meaning we ourselves could not utilize them to drive our IR emitter discussed below in Section 4.0. As a result, we rewrote our servomotor driver code, the shortcomings of which will be addressed later.

In the final implementation of the servomotor driver we used the built-in `digitalWrite` method to place a `HIGH` signal on the control line, delay the processor for the appropriate number of microseconds using the built-in `delayMs` method, using `digitalWrite` again to place the signal pin back to `LOW`. The width of the pulse was calculated using the formula below.

```
PulseWidth = (DesiredAngle * 10) + 550
```

Note, this formula errors on the side of caution and will not allow the servo to reach its full range of motion, achieved by providing a pulse width control system between 500 and 2400 microseconds.The formula creates an signal between between 550 to 2350 microseconds, allowing for the delay in the ISR discussed in Section 4.0.

In our implementation we used relative positioning to allow for ease of use by the operator. The position of the servomotor was changed from its active position to a new relative position based on the position of the X-axis potentiometer mentioned previously. When we sampled the joystick, we used the built-in `map` function to reduce its value in the range of [0, 1023], to a value in the range [-12, 12]. To create a deadzone we would ignore values with an absolute value of 2 or less. The value returned from the joystick was then used to modify the current position of the servo by a number of degrees equal to the value returned by the sampling routine.

4.0 IR Transmitter and Protocol

Finally, we integrated the IR emitter to allow our system to trigger the alarmed receivers provided to us. The communication protocol specification provided by the instructor required that we pulse the IR emitter at 38KHz carrier frequency during the high bits within the transmission of the byte. Transmission of each bit was done by sending pulses of 500 microsecond intervals. To start a transmission, two header bits had to first be sent, a high, followed by a low, then followed by the bits of the byte being sent from LSB to MSB.

To implement this protocol we made use of two timers within the ATmega2560. Timer 3 was used to create interrupts every 500 microseconds, while Timer 1 was used to create the 38KHz carrier frequency on the output pin to the IR emitter. During each interrupt created by Timer 3 the ISR would check if a message was currently being sent, and if so, would change the *TOP* value of Timer 1 appropriately to oscillate create the carrier wave.

To send an 8 bit value from the IR transmitter, there is a *char* variable in the code that can be set. For example, if the letter “A” is set, then the 8-bit ASCII code is sent from the transmitter. When the joystick button is pressed, a boolean flag is set to True. Then, at every interrupt, if that value is still `True`, then one bit is sent. A *int* variable keeps track of which bit in the 8-bit message we are currently sending. If it is the beginning of the message, we first send the 1 and 0 bits to open up communication with the receiver; at every subsequent interrupt, a bit mask is used to pull out the appropriate bit from the 8-bit ASCII code. Once the message is complete, the button pressed boolean is set to False, and the IR transmission is complete.

Various problems occurred when implementing the IR transmitter. Initially, the IR transmitter did not function properly because it was shorted when it was wired incorrectly. It took multiple hours to find this problem as we are not capable of visibly verify the emitter was working. A programming error was also made in the ISR used for transmission. In the bit-masking used, a bitwise OR operation was used instead of the bitwise AND, this mistake was eventually discovered and corrected, enabling our first prototype with communication.

5.0 Load and Optimization

Outside of the timers and interrupts used to communicate with the receiver system, we used the main loop within the Arduino system to sample the joystick, and send the control signal to the servo motor. This approach was used as the limited number of peripherals to sample and tasks to complete did not create a load which required a real-time operating system or a time-triggered architecture.

As this project continues to grow however, a more hands on approach to scheduling will be needed to ensure that tasks are completed as regularly as needed. Phase Two will required a minimum of a time-trigger architecture to ensure that regular events such as that used for the IR transmission protocol, are scheduled such that they all execute as atomic blocks of code.

6.0 Conclusion

Overall, Phase 1 was a success. We implemented a simple scheme to aim an IR transmitter using a servomotor and a joystick, and send data to a receiver using the IR transmission protocol. Once again, as we move forward and add more devices, we will begin to implement a more structured time-triggered architecture. Our code is very modular and will allow this transition to happen easily and efficiently.

7.0 Pictures and Videos

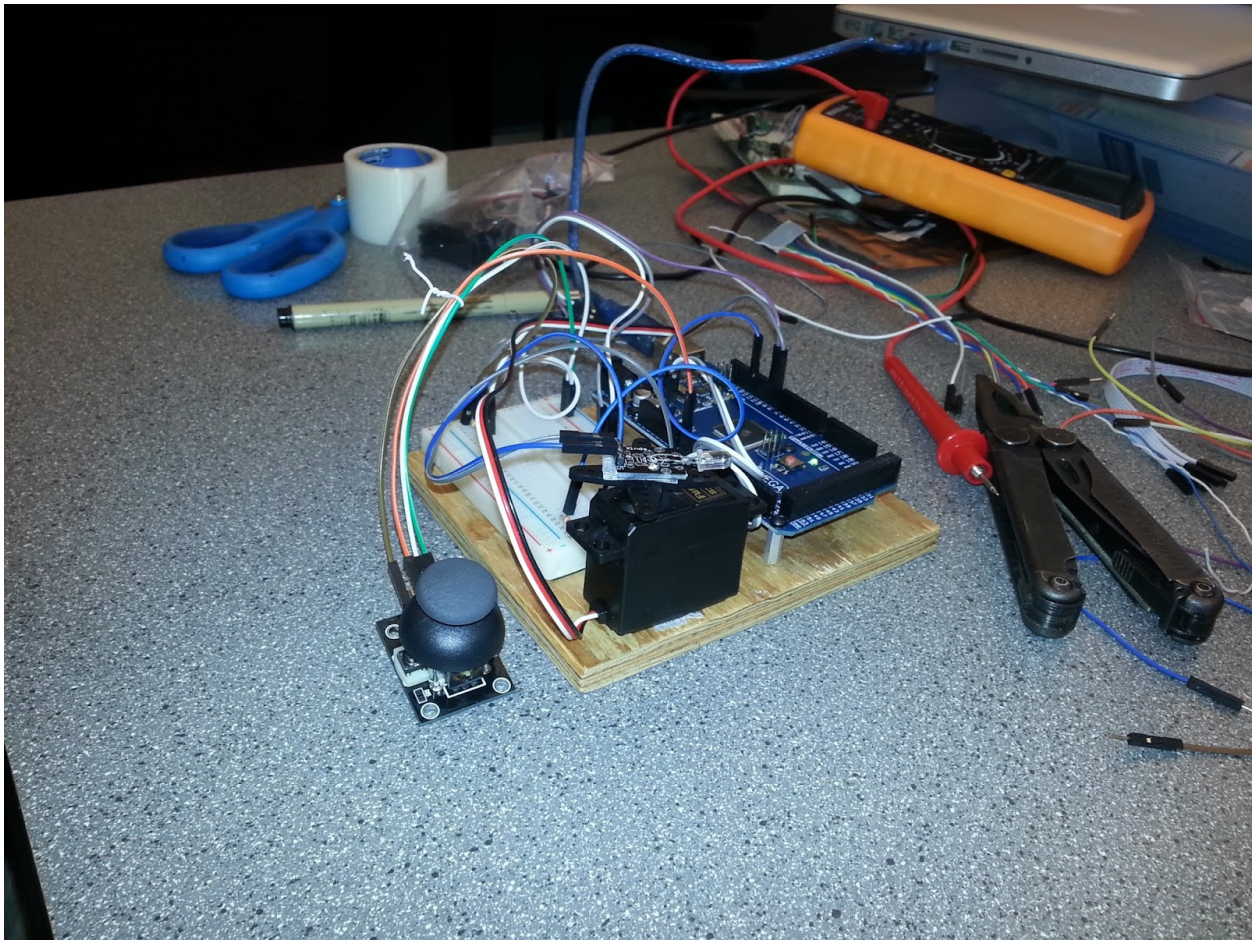

Figure 5 - A side view picture of our implementation

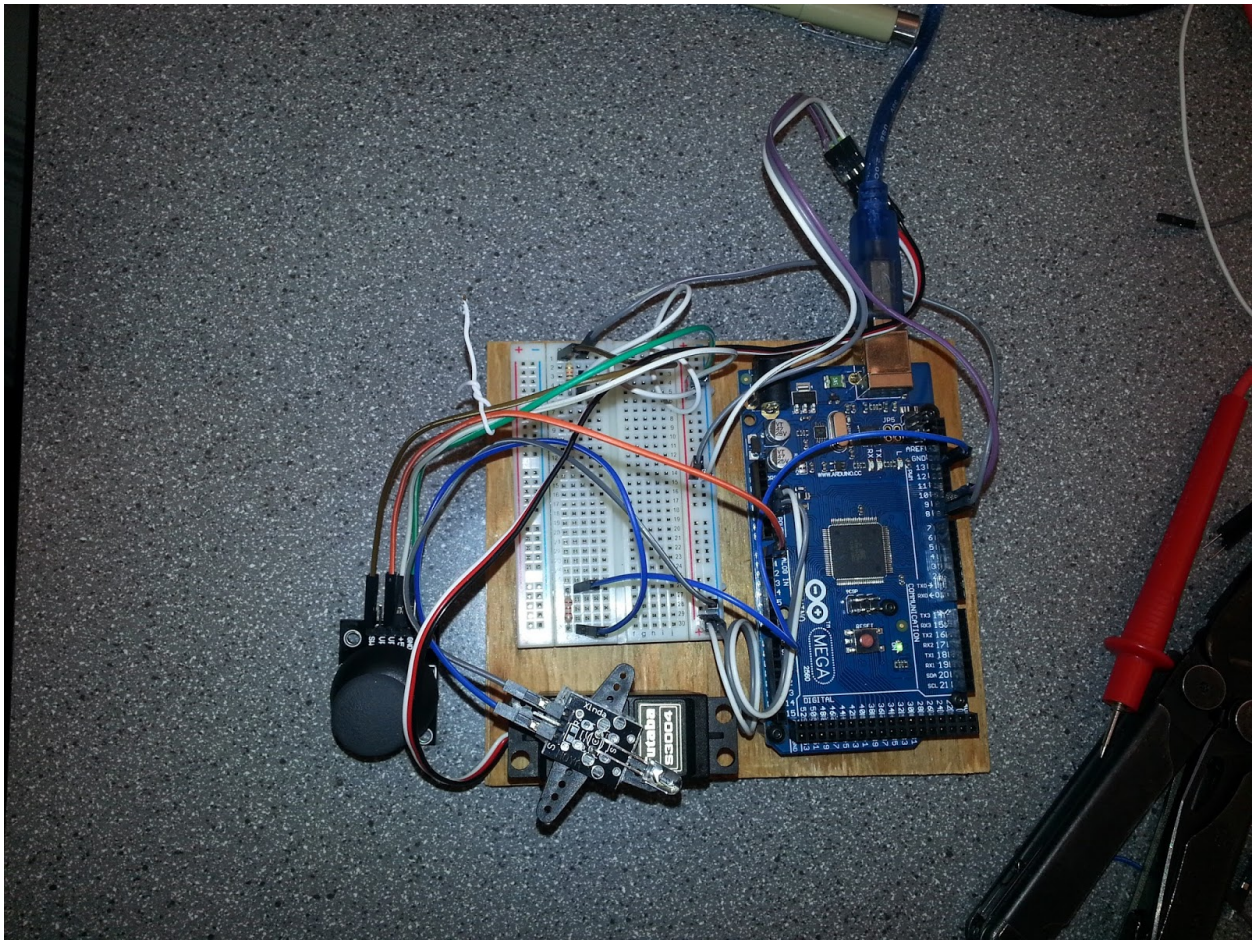

Figure 6 - A top down view of our implementation