



BRADLEY University

Objective-Based Laser Tag with Player-Sense Technology Final Paper

By Sean Morris, Jonathon Jellison, Josh Seelye

Advised by Dr. Aleksander Malinowski

Published May 5, 2019

Table of Contents

Figure Index.....	2
Abstract.....	3
1. Problem Statement.....	4
2. Project Overview.....	6
3. Player-Sense.....	8
a. Sending IR Opcodes	
b. Receiving IR Opcodes	
4. Game Logic.....	11
a. NeoPixel-24 Control	
b. SPI Communication	
5. Communication.....	13
a. Hardware	
b. SPI ESP32	
c. MQTT	
6. Hardware.....	17
a. Flag Hardware Design	
b. Send Circuit Design	
c. Distance Testing	
d. Distance Control-Dynamic	
7. Future Plans.....	30
8. Conclusion.....	31

Figure Index

Figure 1 Flag on Playing Field.....	3
Figure 2 Flag State Diagram.....	4
Figure 3 Project Overview.....	6
Figure 4 Flag Capture Radius.....	7
Figure 5 NEC Communication.....	8
Figure 6 IR Receiver.....	8
Figure 7 NeoPixel-24.....	10
Figure 8 NeoPixel-24 Communication.....	11
Figure 9 Game Logic Flowchart.....	11
Figure 10 Communication Diagram.....	12
Figure 11 Player SPI.....	13
Figure 12 SPI packet	14
Figure 13 Flag SPI.....	14
Figure 14 MQTT diagram.....	15
Figure 15 Flag Communication and Hardware Design.....	16
Figure 16 Initial IR LED send/Receive.....	17
Figure 17 First BJT Circuit.....	18
Figure 18 Function Generator Output.....	18
Figure 19 Vce Overshoot.....	19
Figure 20 Send Circuit Design 2.....	20
Figure 21 Sent/Received Data Packet.....	20
Figure 22 ATmega128 PE0 Pin.....	21
Figure 23 Vce overshoot with ATmega128.....	21
Figure 24 LPF with ATmega128.....	22
Figure 25 Measured Vce with LPF.....	22
Figure 26 final Driver LED circuit.....	23
Figure 27 Distance testing with a potentiometer.....	24
Figure 28 Control-Dynamic Logic Diagram.....	25
Figure 29 D/A Converter and Current Amplifier Circuit.....	26
Figure 30 Full Hardware Design.....	27
Figure 31 ATmega128 and ATmega168 Pinout Diagram.....	28
Figure 32 LCD Display.....	29

Abstract

Laser tag is a widely known combat simulation game, where opposing teams use beams of light to shoot fake bullets at each other to eliminate enemy players from the game, all while trying to complete an objective to win. Each laser tag match can have different game modes (objectives) ranging from eliminating all enemy players to other in-depth game modes such as domination and bomb squad, where players are required to interact physically with devices to complete an objective.

While current laser tag systems are fun, and the variety of game modes available provide a more dynamic experience for each player, they lack player-to-player communication and autonomy within objectives. Current laser tag systems have no way of communicating in-game statistics to players, such as team points, current kills, current deaths, and remaining ammo, forcing players to advance through the match without knowing their team's current rank against the others. In addition, game modes like domination require players to press and hold down a button for 10 seconds to capture a flag. This forces them to put down their weapon and divert their attention away from the battlefield, leaving them out in the open to be eliminated.

By designing a communication structure between all players and objectives, and introducing the ability for objectives to "see" players with Player-Sense technology, effectively eliminates the problem of player-to-player communication and autonomy within objectives. This makes for more fun, in-depth, and realistic laser tag experience.

1. Problem Statement

Laser tag, while being a fun game to play, is heavily aimed at the commercial industry, and not well built for home use. This forces consumers to pay a hefty price for a simple hour of high-tech fun. Commercial laser tag companies normally have a variety of game modes; Capture the flag being one of the most common and well known. It is played by placing an electronic flag (objective) in the center of the playing field (map) shown in figure 1. Players from opposing teams (Red and Blue) will come up to the flag and must press and hold a button for 5 or 10 seconds to capture it depending on the state of the flag (see state diagram below). Once captured, the flag's lights turn to the appropriate team's color, signaling that the flag is now owned by that team. The game then keeps track of which team owned the flag for the most amount of time to determine the winner.

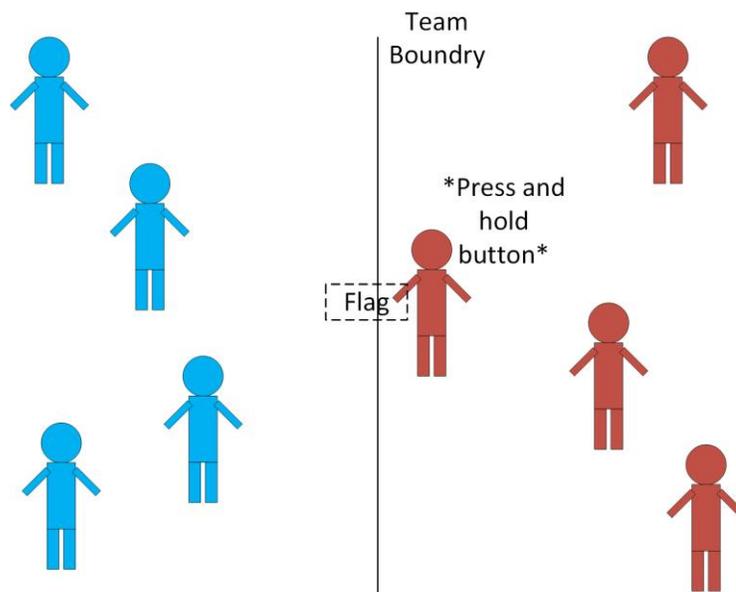


Figure 1 Flag on Playing Field

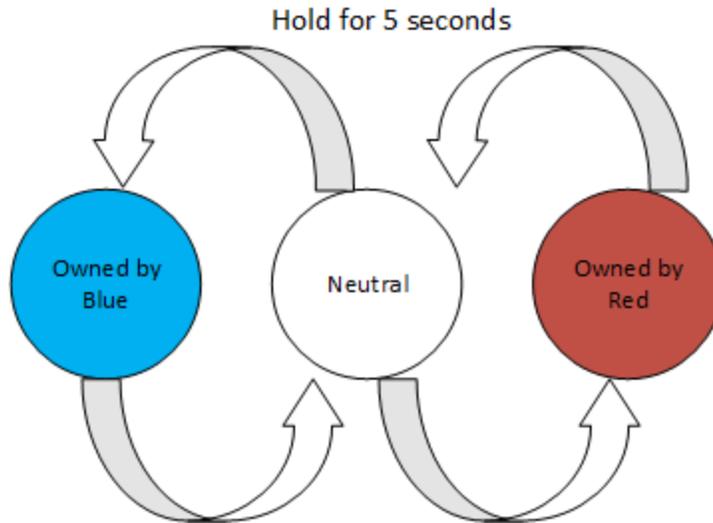


Figure 2 Flag State Diagram

Despite this being a fun and interactive game, there are many problems with this configuration. For starters, this is only available in the commercial industry. This requires people to spend money anytime they want to play a game of laser tag. Presently, a game mode like this, and an in-depth laser tag experience, does not exist in the home market. There is only one button available to press for capturing, which makes it hard to capture the flag, considering the entire enemy team is always staring down their sights at the flag. Having a button attached to the objective forces the players to physically interact with the button and take their eyes and aim off of the battlefield. This eliminates the autonomy of objectives, meaning that players have to interact physically with the flag to change its state.

Laser tag in itself is not problem free either. At any given moment throughout the match, players on both sides have no way of knowing which team is winning, how many points their team has, or how they are doing as a player to support their team. Knowing these critical pieces of information is important during a match because it helps you to know if you can take a break from playing offense, or if your team needs all hands-on-deck. It brings team morale down, as well as makes the overall experience less fun.

2. Project Overview

In order to combat the aforementioned problems, a new laser tag system needed to be built. This laser tag system would be communicating all in-game statistics to every player in real-time, including ammo, team points, eliminations by the player, how many times the player was eliminated, flag statuses, and even health.

On top of the communication scheme, a way of developing autonomous objectives needed to be implemented so that objectives can change their own states without players physically interacting with them. In order to do this, a capture radius needed to be established, as shown in figure 4. With this implementation, players do not have to touch any part of the flag to capture it, and there can be multiple players from each team within a capture area, removing the problem of having all eyes on one person as he or she captures the flag. As more people from the same team enter the capture zone, the time it takes to capture the flag fully (or change the flag's state) goes down. If there are an equal number of Red and Blue players inside the capture zone, the flag is considered contested; it is in a frozen state such that neither team captures or loses the flag until one of the players is eliminated.

All of this information was to be displayed on a touchscreen LCD that would be mounted to the back of the laser gun for ease of use by the player. The gun would also have a speaker attached to it so that it could alert the player if they were shot/eliminated, provide game updates (as if a squad leader was speaking to them), as well as make laser sounds when the trigger was pulled.

Figure 3 is a high-level description of the flag, gun, and communication between the two. The flag will consist of 2 MCU's, flag indicator lights for players to know the status of the flag, player sense hardware, and the Wi-Fi module to communicate to other players via a player's home Wi-Fi network. The player will have the ability to adjust the player sense hardware to make the capture distance of the flag vary to their preference. The gun will consist of 2 MCU's, a Wi-Fi module, IR communication, and multiple peripherals for both user input and data output via a speaker, RGB LEDs, and an LCD. The LCD was to be the main source for in-game statistics of current objectives and player information during the game. This would include objective status, team points, player points, and ammo count. The MCU of choice for the flag and gun will be an ATmega128 or ATmega168. Communication between the flag and the gun will be via the Wi-Fi controllers of an ESP32, located both on the flag and the gun, and the home network of the player's house.

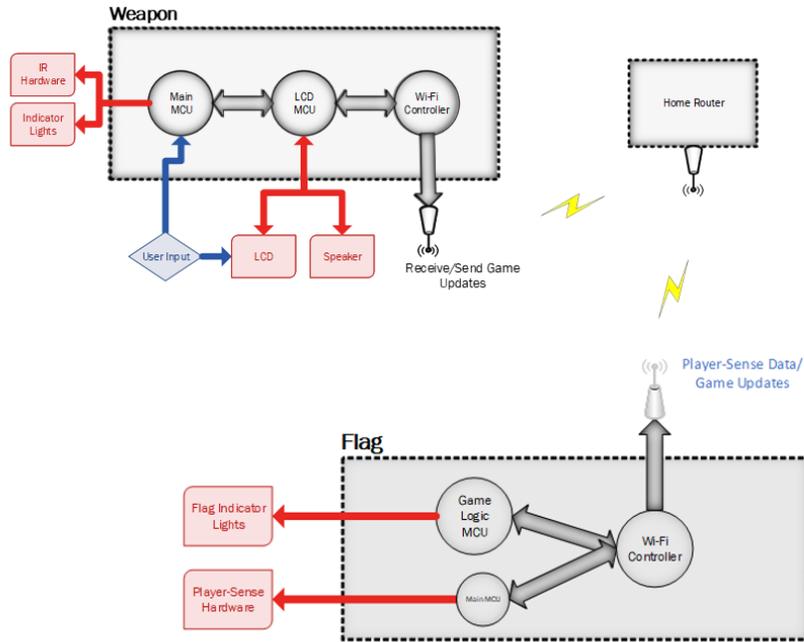


Figure 3 Full Project Overview

All of these systems would be packaged into a convenient on-the-shelf product that would be available for purchase by the public, bringing the fun of laser tag and high-tech game modes to homes at an affordable price.

3. Player-Sense

The capture radius of the flag is able to be changed within a range from 3 feet to 15 feet so that the player can adapt their flag to their own surroundings. Player-Sense is the technology that allows the objective(s) to see a player within a said specified radius. The flag will emit a unique opcode using infrared (IR) light in 360° from the center of the flag to create the capture radius, as shown in figure 4. When the player(s) receive this opcode, the player transmits through the communication system that they are currently present inside of the specific flag's radius. From there, the flag is able to register that someone is within its radius, and can begin to capture for that team.

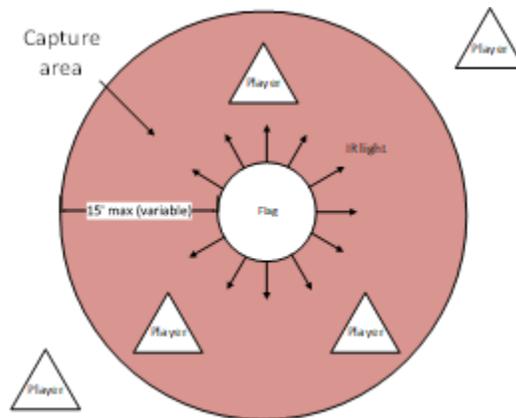


Figure 4 Flag Capture Radius

A. Sending IR Opcodes

Sending an opcode via IR was accomplished by using the NEC protocol, which is an industrial communication standard for sending and receiving IR opcodes. An example of how NEC protocol works is shown below in figure 5. The ATmega128 was chosen to run the flag firmware and do most of the computations relating to IR data processing. Timer 0, an 8-bit timer, was used to generate a 56 kHz square wave using the fast PWM mode on the timer. Timer 1, a 16-bit timer, was used to time the spacings shown in figure 5 using the clear timer on compare match (CTC) mode, and determine when timer 0 would output its waveform to the microcontroller (MCU) pin.

An opcode is sent by calling the *send_byte(uint8_t data)* function, which sends *data* to a global variable named *ir_data* and loads the output compare registers (OCR) of timer 1 for the start bit shown in figure 6. When the contents of timer 1 have reached either of the OCR registers, an interrupt is triggered, causing the interrupt service routine (ISR) to turn on/off the output of timer 0 to the MCU pin, as well as move a mask bit moves across the value of *ir_data* determining a 1 or a 0 for each bit. Then, depending on the value of the current bit, the output compares registers (OCR) of timer 1 are updated with the appropriate on and off times in figure 5.

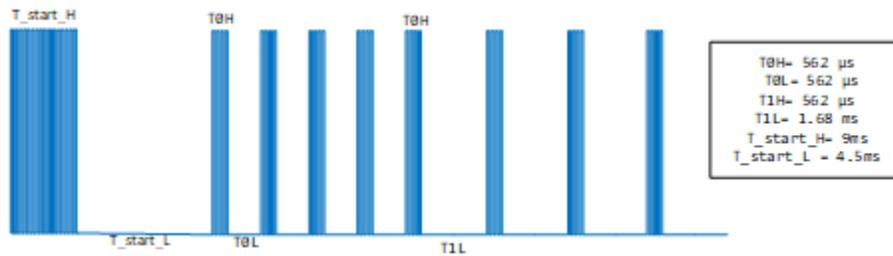


Figure 5 NEC Communication

B. Receiving IR Opcodes

Receiving an opcode via IR was accomplished using the same NEC protocol. The ATmega128 was chosen to run the player firmware, and do all of the computations relating to IR data processing. Timer 1, a 16-bit timer, was used to determine the spacing between each falling edge of the incoming signal from the IR receiver (figure 6). This was done by utilizing the input capture unit on timer 1. The IR receiver takes the incoming pulsed data from the flag and turns it into a simple digital signal that the MCU could easily process.

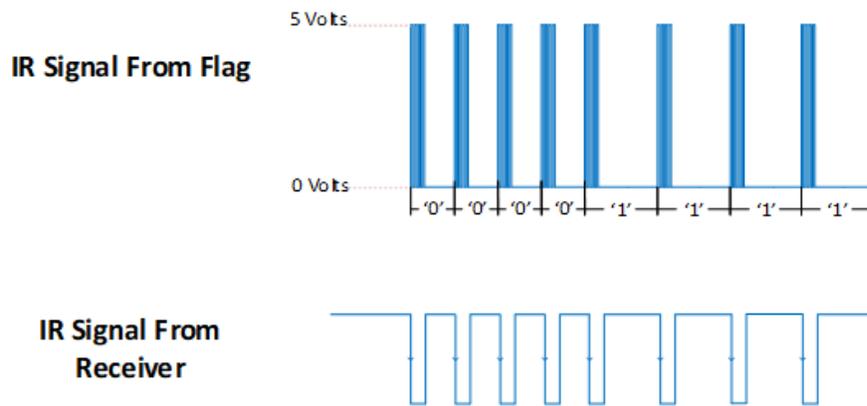


Figure 6 IR Receiver

A state machine was used to determine what opcode was being transmitted through the IR data packet. While writing this portion of the code, there were a lot of challenges that were faced. For one, figuring out a way to understand what was happening behind the scenes of the code was difficult. A debug library was written that provided useful debug statements to be used in code that would print out variables and strings to the serial terminal on a computer. This was done using the terminal emulator program called PuTTY, and USART unit onboard the ATmega128. Once the debug library was complete, seeing inside the code was easier. However, the state machine would constantly become “off by one state” at random periods during transmission. By flipping

pins on and off, and using the debug library, it was found that the timer was not being properly reset at the end of a state, therefore causing timing issues between states.

4. Game Logic

For the flag to be able to change its lights based on its current state and the number of players inside of its capture radius, there had to be some sort of logic that controlled the flow of the game. Originally, it was thought that all of the game logic would be written on the ATmega128. However due to the timing constraints of the NeoPixel-24, a faster clock frequency was required, and so the ATmega168 using a 20 MHz crystal was chosen to be the game logic MCU.

A. NeoPixel-24 Control

This specific device, shown in figure 7, was chosen to be the flag indicator lights due to the cost, ease of use, and the amount of light customization available. The NeoPixel-24 communicates via 1-wire high-speed PWM to transmit all of the color data. Figure 8 shows how each bit is sent over the communication wire, as well as how a color packet is formed. In order to control this LED ring, a library was developed consisting of various functions that altered a 24-element array. The array would be changed accordingly by the game logic interrupt process, and then the array would be sent to the ring at the end of the process. Computing the game logic and updating the ring took about 5 ms. Note, sending the individual bits were written in assembly because of the necessary accuracy of timing.



Figure 7 NeoPixel-24

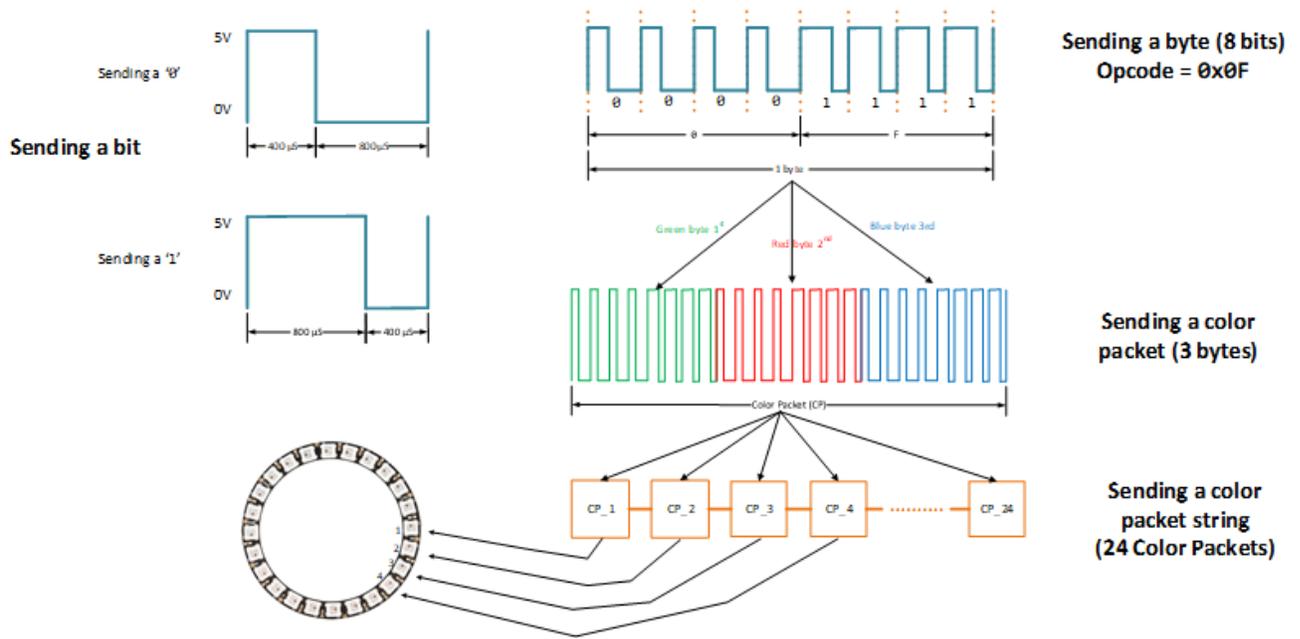


Figure 8 NeoPixel-24 Communication

B. SPI Communication

In order for the game logic to make decisions based on the number of players, there needed to be communication from the rest of the system. In order to do this, SPI communication was implemented between the ESP-32 and the ATmega168. Asynchronously, the SPI information is received from the ESP-32. The *flag* struct is updated with the appropriate information, where it is then used by the game logic process to update the LED ring. Figure 9 below shows this process.

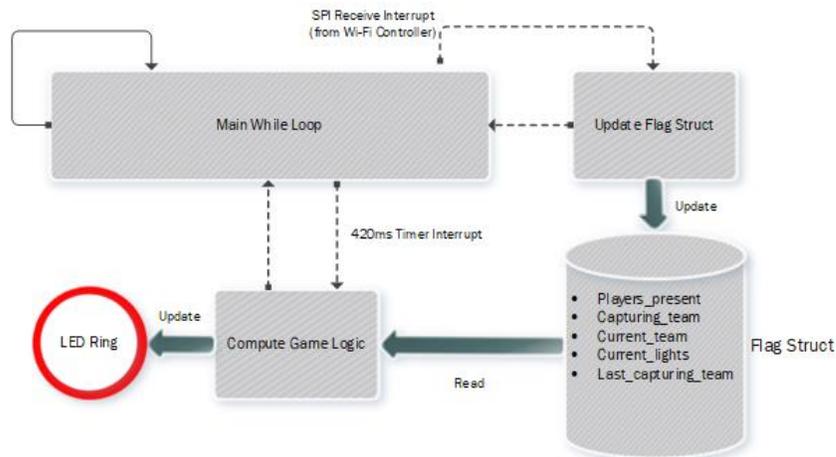


Figure 9 Game Logic Flowchart

5. Communication

To be able to correctly communicate all of the logic between all of the systems, a communication system needed to be developed. This way, each component would know what its current state is and what it is waiting on. In order to be able to achieve this, certain hardware and communication standards needed to be settled on. A figure was created to handle all of the challenges and show the intercommunication between the whole system; this figure is shown below in figure 10.

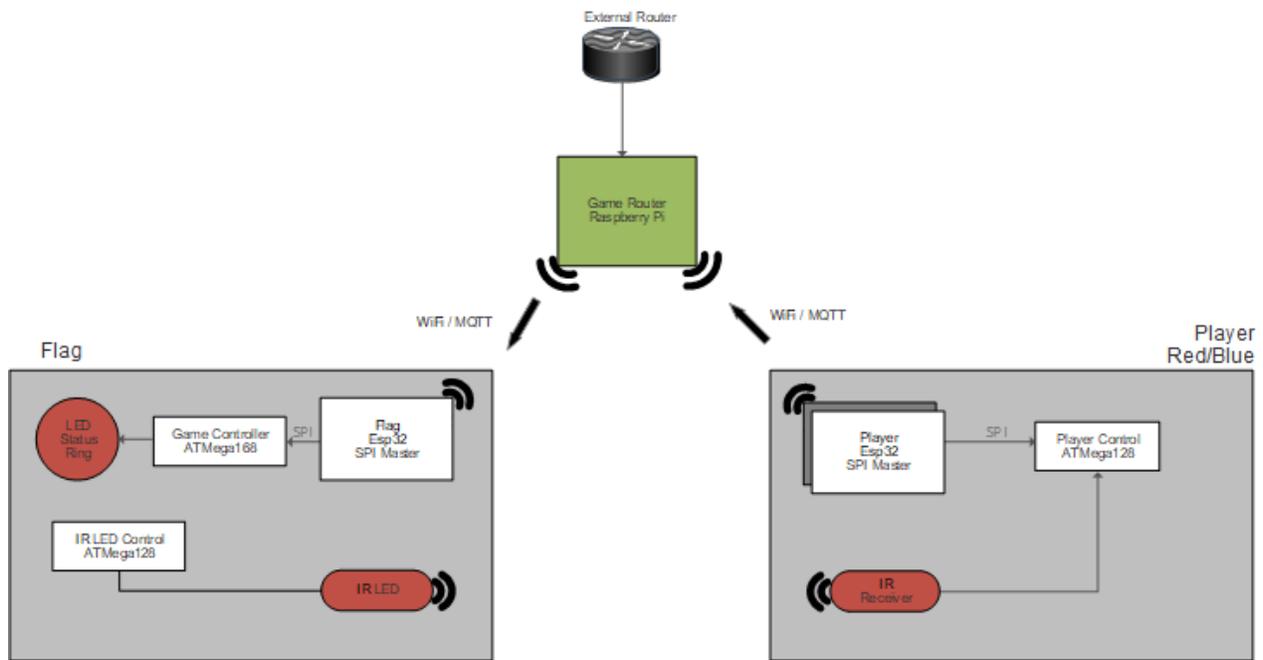


Figure 10 Communication Diagram

A. Hardware

To achieve the communication diagram above a few more devices needed to be selected to handle each of the challenges. An access point to handle all of the wireless communication and a device to transmit information between the player and the flag wirelessly that is also capable of communicating with other MCUs.

- **Raspberry Pi**

To fill the position of the access point in the system, the Raspberry Pi was selected to do the task. This device was selected because it was able to be set up as an access point and also could connect the network to the internet. The importance behind being able to connect to the internet would be for firmware updates. This allows the system to be able to distribute firmware updates to players and the flag.

- **ESP32**

For the Wi-Fi portion of the communication diagram, a board needed to be selected that was able to connect to the access point while also being able to communicate with other MCUs. The device that was found to fit this position best was the ESP32. The ESP32 was decided on for a few reasons: one being that its clock speed can run at 240 MHz for much faster processing than its counterpart the ESP8266, two it has innately faster Wi-Fi its counterpart, and finally three it has 48 GPIO pins which is enough to complete all the tasks required for inter-device communication

B. SPI ESP32

While looking back at figure 10, it can be seen that SPI was chosen as the main way to communicate between devices. In this case, it will be used to communicate on the players end between the infrared ATmega128 and the players ESP32(shown in figure 11) and on the flags end between the game controlling ATmega168 and the flags ESP32(shown in figure 13).

- **SPI Player**

SPI on the players ended up being a tricky task to deal with. From figure 11, it can be noted that the SPI line is heading toward the ATmega128. This is to show that SPI, in this case, is being directed by the ESP32 instead of the ATmega128. This is because the ESP32 cannot be set up in SPI slave mode and because of this needed to be set up in SPI master mode where it commanded all of the data transactions. Ideally, the ATmega128 would send the information to the ESP32 immediately after it received it from the IR receiver. In order to solve this issue, the ESP32 on the player end was set up to poll the ATmega128. The poll rate was set up for 50 ms to allow for at least one pulse of the flags IR Opcode to send which took around 40 ms.

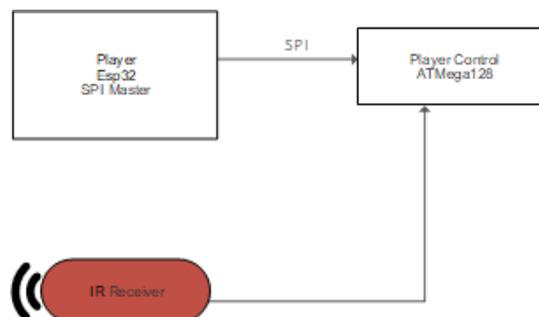


Figure 11 Player SPI

- **SPI Flag**

On the flag end, SPI was also used for the inter-device communication between the flags ESP32 and the flags ATmega168. In this case, the ESP32 needed to be the master communication could work according to plan. The ATmega168 needs specific information from the ESP32 for the game logic to correctly work. To solve this, an SPI packet was designed to handle all of the information which is shown below in figure 12. In the first byte, the first bit was set to be the read or the write bit. If that bit was a 0, it would tell the ATmega168 that it wants to read from the selected variable and if it was a 1, it would say that it wants to write to the variable. The next seven bits are reserved for variable select to selection what variable SPI is going to write or read from. In the next byte, it needed to be decided how many teams could exist and how many players can be on each team. It was decided to split the byte down the middle to allow there to be up to 15 teams and allow for up to 15 players on each team. The byte of 0xff was also reserved for the game state of contested.

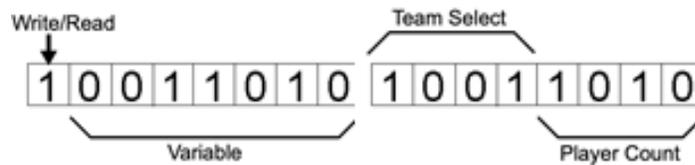


Figure 12 SPI packet

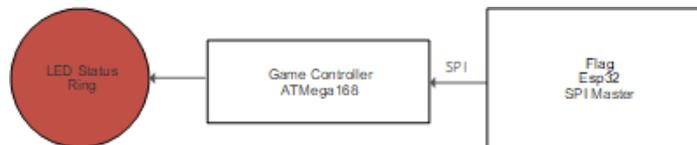


Figure 13 Flag SPI

C. MQTT

To implement the Wi-Fi communication, it was decided that the best protocol to use was TCP. This was decided so that packets would never be lost during

transmission and it would make sure that all of the devices would always be up to date if they were within the range of the access point. It was found that other protocols have been developed on top of TCP and solved everything that was needed to achieve the communication diagram. The protocol that was decided on was MQTT or mosquito. Mosquito is a publish and subscribe based protocol, and a full diagram of how MQTT works is shown in figure 14. MQTT has what is called topics which is essentially just a string identifier for the “chatroom” that the device is talking in. Whenever a topic is created any of the devices that are connected to the broker or in this case, the access point can subscribe to that topic. This means that whenever a message is sent in that topic, any device that is subscribed will receive that message. To send a message in that topic, any of the subscribers plus any other device can do so. This is performed by “publishing,” which sends the message off to the broker to handle all of the distribution.

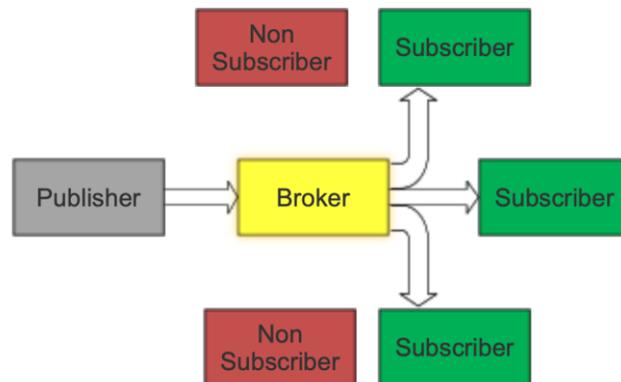


Figure 14 MQTT Diagram

For the use in this project, there was a topic set up for each and every flag. This allowed for communication to occur just between everyone who subscribed to the flags and the flag itself. For the subscribers, it was decided that all that needed to be subscribed to each of the flags topics is the all of the players and of course the flag itself. From this, each of the players would be updated with important information about each of the flags separately while still being able to distinguish where each of the messages is coming from.

6. Hardware

In order to implement game communication and low level-logic control, sending/receiving circuitry had to be designed to send/receive the data packets of the flag and the gun via IR communication. This circuitry had to be made to send a signal at 56 kHz to a certain radius away from the flag to create a “capture zone.” When players enter this capture zone, IR receivers on the players’ vests will pick up this signal and identify the data packet within the 56kHz signal for instructions as to what actions need to be taken for the player and flag statuses. Below will be a summary of this design process and all technical specifications of the circuitry design.

A. Flag Hardware Design

In order to accomplish the flag hardware design, the following goals had to be accomplished. First, send/receive circuitry had to be made that would allow for an MCU to control a 56kHz signal being sent to an IR LED for sending. This circuitry would have to be designed to handle any size data packet, have built-in safety features, and be low power for the eventual use of batteries to power this system. Two, determine ways to vary the capture zone of the flag while keeping in mind cost, efficiency, and maximum amount of control for the user benefit; I.E., allow the player to choose any size capture zone they wanted up to a 25-foot radius from the flag. Lastly, as mentioned in task 1, determine final power calculations for the eventual implementation of batteries into the system for portability. The battery life of the system needs to have a minimum battery life of 1 hour. The following figure below shows how the hardware aspect of this project plays into the entire design of the flag.

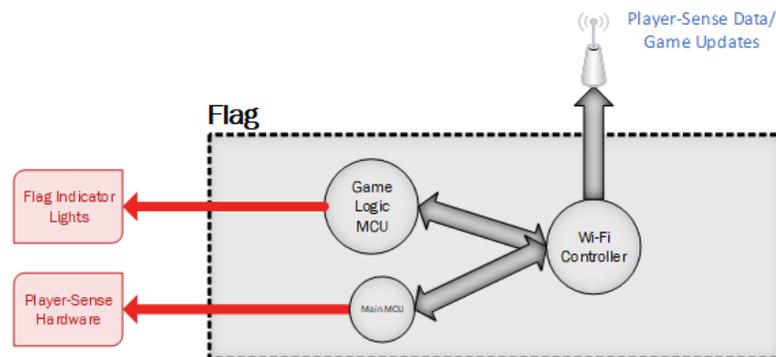


Figure 15. Flag Communication and Hardware Design

In the above figure 15, the flag indicator lights and player-sense hardware is where this work ties into the flag design.

For this design, the following IR LED, SFH 4546 was chosen. Other IR LED's in the same family were used, such as the SFH 4547, and SFH4545. However, the SFH 4546 was chosen for its Viewing angle, forward voltage, and forward current parameters. With a viewing angle of 40° ($\pm 5^{\circ}$), this would allow us to potentially only need 9 LED's to get 360° of coverage on the x-, y-, and z-axes. To go along with this, the LED's had a forward voltage range of 1.3-5 volts, and a forward current limit of .100 A. Using the circuit designed in the preliminary research portion last year for this project, it was determined that this LED and 70 mA had a sensitivity range of over 30 feet, the length of the laboratory. That sent and received signal is shown in the below oscilloscope image.

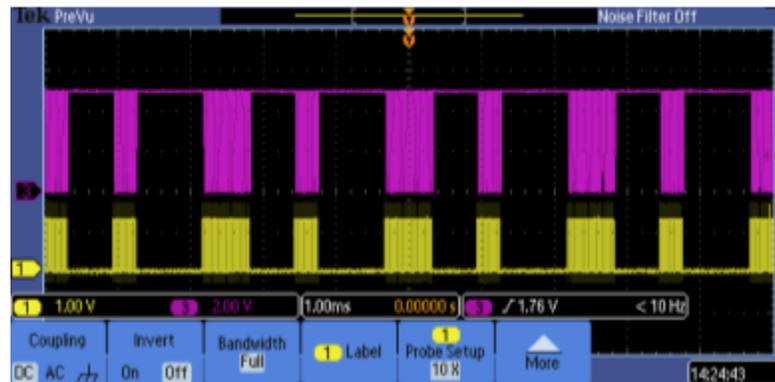


Figure 16 Initial IR LED Send/Receive

In the above image, passive IR receivers were used. Active IR receivers were used for the rest of the project. Channel 1 shows the sent data signal, while channel 3 is the received signal on the opposite side of the lab.

For the IR receivers, the TSOP4440 active IR receiver was chosen. It had a carrier frequency of 56kHz, as needed to detect the MCU data packets and a wavelength of 950 nm.

B. Send Circuit Design

For the send circuit design, a simple BJT circuit was created. At the beginning of the semester, the send data packet code had not been completed yet, and so a function generator was used in place of an MCU output. A 2N2221A NPN BJT was chosen due to their availability in the lab. The following circuit design was

made in figure 17. This circuit consists of a R_c of 30Ω , R_b of $1.2k\Omega$, V_s of $5V$, and V_b being the function generator input. A beta value of 30 was used for the gain.

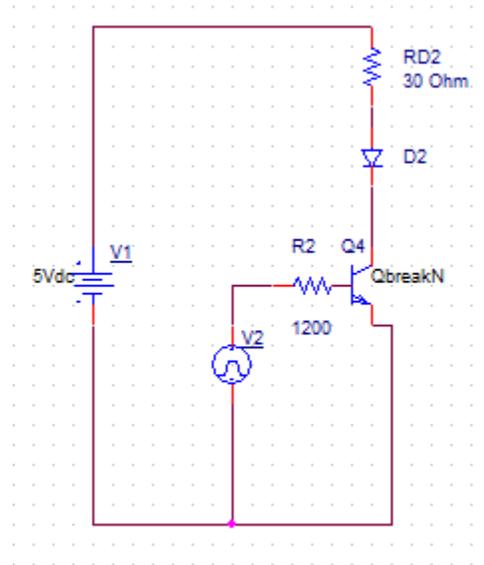


Figure 17 First BJT Circuit

This circuit proved well as it was able to send and receive a simple 50% duty cycle 56kHz pulse from one side of the lab to the other. The only issue that was found in this design was in the output of the function generator. A decent amount of noise was found on the output and was being amplified by the BJT due to its amplification characteristics. In some cases, an overshoot of 7V was observed across V_{ce} of the above circuit. Figure 18 shows how this noise caused overshoot on the logic level high of the 50% duty cycle signal.

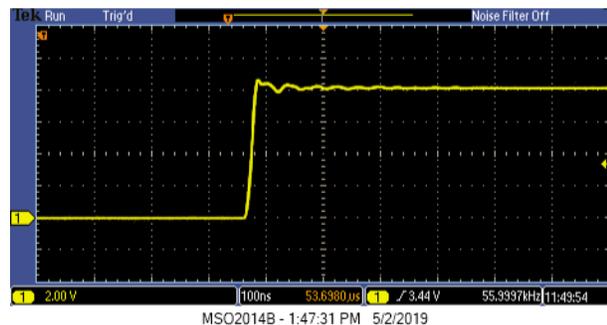


Figure 18 Function Generator Output

While the overshoot present in the above figure is minuscule, it was still enough to cause such a large overshoot over V_{ce} .

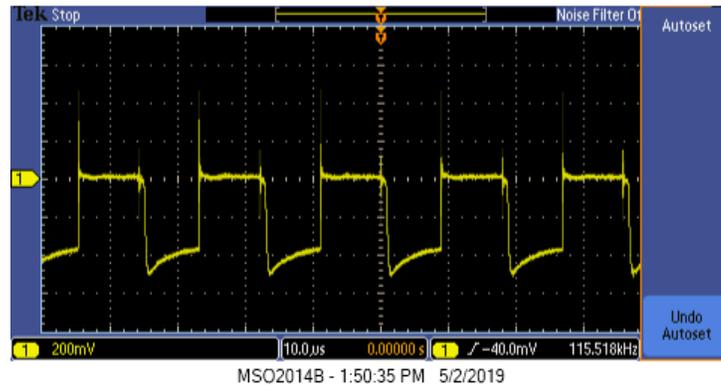


Figure 19 V_{ce} Overshoot

As shown in figure 19, there is a considerable amount of overshoot present across V_{ce} of the BJT circuit. Note that the function generator output was set to a V_{pp} of 5V with it being from 0-5V. After talking to Dr. Huggins, it was determined that the overshoot was simply due to an artifact, inductance, from somewhere within the function generator and was thought that after moving on to V_b being the MCU instead of the function generator that the artifacts will be gone. However, what was an interesting observation was just how little noise would correlate into such a high amount of overshoot. In any case, the overshoot, for now, is ignored.

Send circuit design 2 now incorporated the MCU as V_b . Using the same circuit design as above only with this one change, the new NPN circuit was able to send flag opcode to an active IR receiver on the other side of the lab. Note that at this time in the design process, the send opcode was completed. Below is the updated circuit model with the sent and received data.

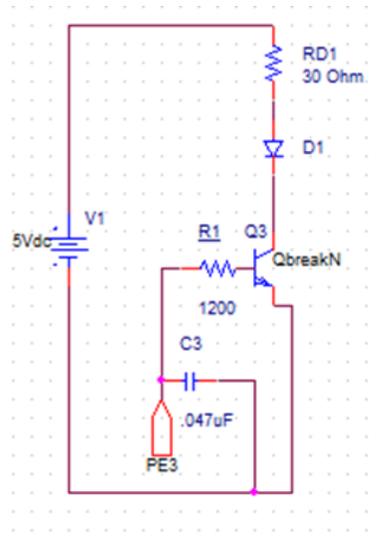


Figure 20 Send Circuit Design 2

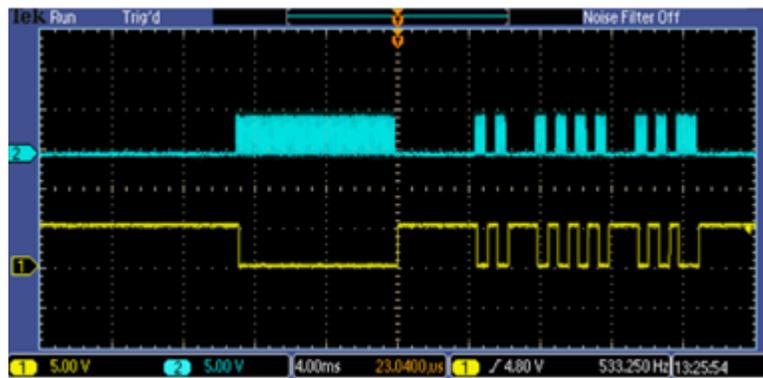


Figure 21 Sent/Received Data Packet

In figure 21 channel; 1 is the output of the MCU that feeds into PE0 of the above circuit diagram. Channel 2 is the receive IR signal. As shown, the received signal is identical to the sent signal proving that this circuit design works.

However, the same overshoot issue, as in figure 22, again became an issue when observing V_{ce} . This time it was very apparent that the inductance seen on the pin was coming from the MCU and not the circuit itself. Figure 22 below is taken on PE0 of the ATmega128 in charge of sending the data packets.

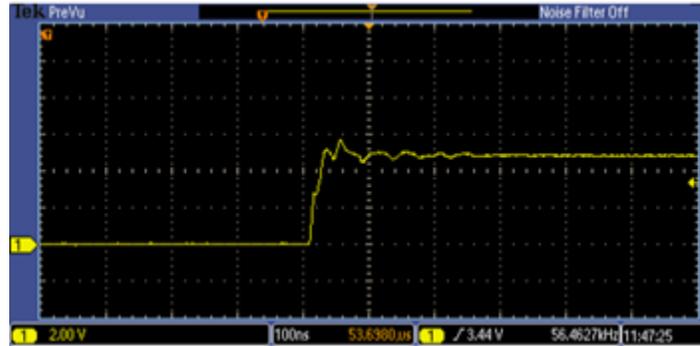


Figure 22 ATmega128 PE0 Pin

It is obvious that the impedance that is unwanted in figure 22. It is not uncommon for an MCU pin to require additional circuitry for filtering purposes. Figure 23 shows the effects of the overshoot on V_{ce} . While the 5V overshoot only lasts for a short period of time, the issue lies in the datasheet of the IR LED only allowing for a forward voltage of 5V. The LED was still turning on and sending the requested data, but the life of that LED may be limited due to the 5V of overshoot. This issue had to be solved before moving further in the design process.

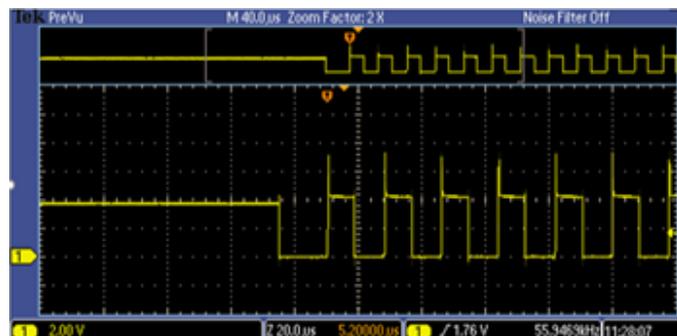


Figure 23 Vce Overshoot with ATmega128

In order to solve this issue, a simple low pass filter was needed. It was made using the $1.2\text{k}\Omega$ resistor, or R_b and a capacitor of $.047\mu\text{F}$. Once the LPF was implemented in series with PE0 of the ATmega128, the overshoot was immediately reduced, as shown in figure 24.



Figure 24 LPF with ATmega128

Figure 25 also shows how the overshoot across V_{ce} was also reduced due to the LPF.

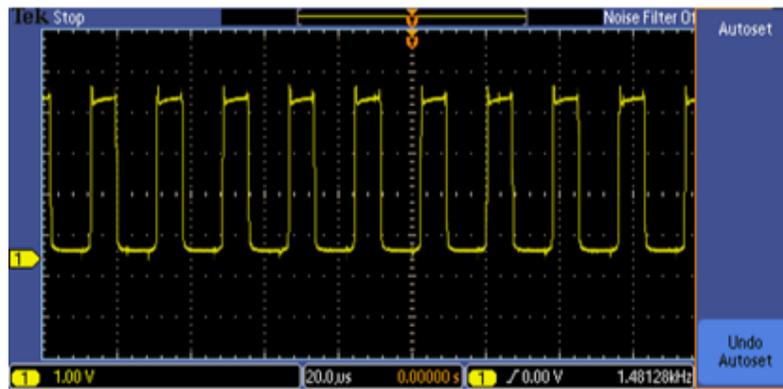


Figure 25 Measured V_{ce} with LPF

The final step was verifying the FOV of the IR LEDs and determining the amount of IR LEDs need to get 360 degrees of coverage on the x-, y-, and z-axes. After doing simple measurements by turning the circuit until the IR receiver stopped seeing the incoming data packet, it was determined that only 8 IR LEDs would be needed for the implementation of the flag. Whether this was due to the LEDs having a FOV of $\pm 5^\circ$ or the signal bouncing off objects, only need 8 LEDs instead of 9 meant less power would be needed to run the system. Figure 26 now shows the final circuitry with the implementation of the LPF and the 8 IR LEDs.

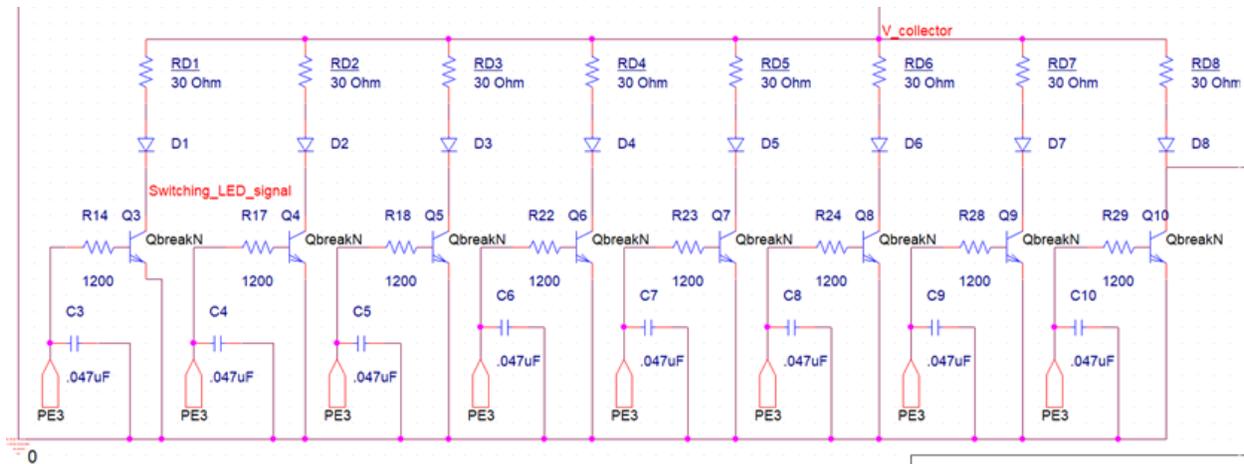


Figure 26 Final Driver LED Circuit

C. Distance Testing

The next phase was to determine a way of limiting the radius of the flags capture zone. The first step to this process was determining a way of taking my distance measurements. To do so, an oscilloscope, voltage source, and sending circuitry were placed onto a movable cart. The tape was then used to map out a grid pattern in the senior lab to 3 feet intervals.

At this point in the flag design process, the code was finished being written for receiving and decoding incoming IR data packets. Uart was set up with the ATmega128, and through PuTTY, the transmitted data packets were able to be watched as they were received and decoded.

The first attempt at varying the distance of the flag was simply by adding my resistance to R_c to restrict current flow to the IR LED. A 500Ω potentiometer was attached between V_s and R_c in series. With the potentiometer turned to zero the cart was pulled back to the 25 ft mark away from the IR receiver. This was the maximum distance that a player would be able to set the flag radius. I_{max} of this test was set at 70mA to ensure the 100mA max of the IR LED was never met, and possibly harming the IR LED.

After turning the potentiometer dial to the point where the IR receiver stopped seeing the sent data packet, the resistance of the potentiometer was recorded. After repeating this step for 3ft intervals, the following data was taken and displayed in figure 27.

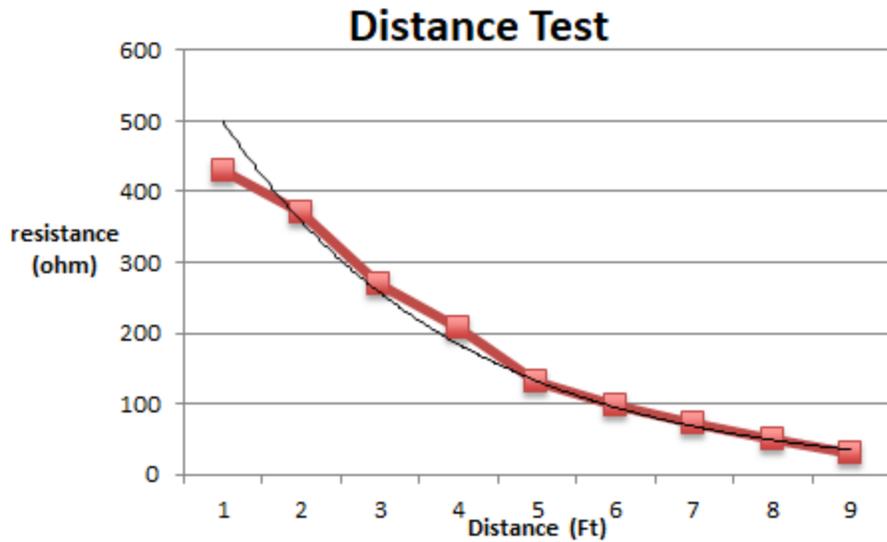


Figure 27 Distance Testing with a Potentiometer

With Voltage remaining at 5V, the inverse relation is obvious and proven by figure 27. With the data following this relationship, it was proved that a player could vary the capture distance of the flag using a potentiometer from placing a dial on the flag for the player to move to set the capture distance.

While this solution works for varying the capture distance, an objective of the design of the flag was for it to be able to vary the capture distance via an MCU by having the player simply enter on a keypad the exact amount of feet they want for the capture distance. This will be explained in detail next.

D. Distance Control-Dynamic

The hardware logic diagram for how the ATmega128 would control the capture distance is explained in figure 28.

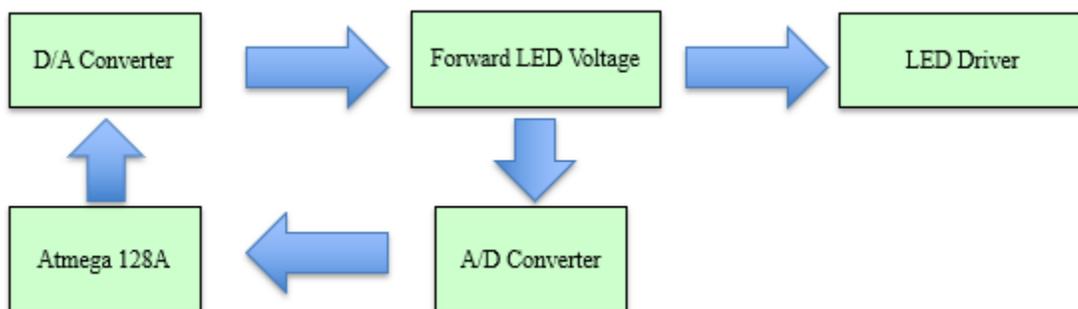


Figure 28 Control-Dynamic Logic Diagram

From Figure 28 a D/A converter was used to control the forward voltage, rather than the forward current of the IR LED to change their output intensity and thus vary their capture distances. From there a feedback loop, going back into the ATmega128 through its A/D converter, will monitor the nodal forward voltage and adjust for any variations in it to maintain the capture radius the player wants.

Unfortunately, the D/A converter on the ATmega128 was not able to be used, and an IC 8-bit input D/A converter had to be used instead. This D/A converter would also require an opamp to act as a current to voltage converter and amplify the output voltage of the D/A converter.

Professor Gutschlag already had an IC and circuit diagram for me to follow and reference as a starting point for this design.

A current amplifier circuit was also needed to make sure each IR LED still had enough current to turn it on. A Darlington pair of BJTs were used coupled with another LPF to ensure the output did not have overshoot. Figure 29 shows the D/A converter circuit with the current amplifier. This portion of the project would not have been possible without Professor Gutschlag

In initial testing of seeing how distance could be controlled by varying the forward voltage, it was clear that minuscule changes in the mV, of the forward voltage, was enough to change several feet of capture distance using the same distance test with the potentiometer. This meant that a high resolution was needed to control the forward nodal voltage of the IR LEDs. The 8-bit IC D/A converted being used would have enough resolution to accomplish this goal of control.

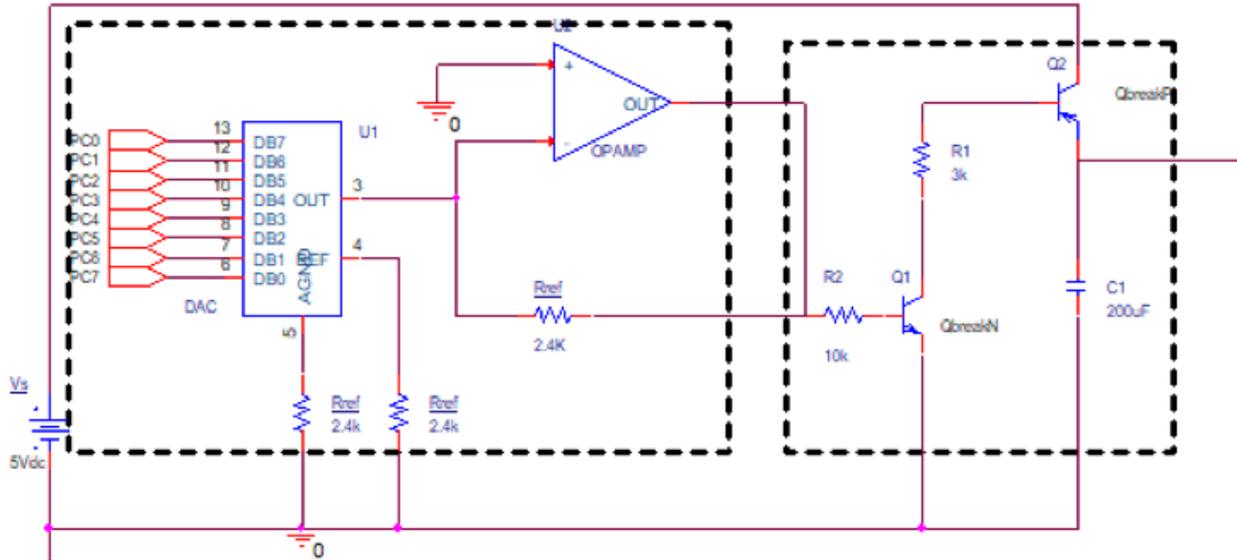


Figure 29 D/A Converter and Current Amplifier Circuit.

A 200µF cap was used to limit oscillation on the output of the current amplifier, but with further testing, the cap could potentially be reduced. Note that there was not enough time to create code for the ATmega128 to send 8-bit inputs to the D/A converter. This was solved by creating a pull-up resistor tied to dip switches to toggle different 8-bit 5V inputs into the ATmega128. This would only be temporary as the final product of the flag would have the D/A converter accept inputs from the ATmega128 and is fine for testing.

From here, the feedback control part of the logic design was created. The feedback control would take a voltage in parallel with the IR LEDs and monitor it and compare it to what the voltage should be for a certain the A/D converter on the ATmega128 would never be able to handle current across the IR LEDs. Therefore, a current to voltage converter was needed to. Unfortunately, this portion of the flag did not work. Also, the code was never designed to accept inputs from the A/D converter.

Figure 30 shows the circuit design for the A/D converter implemented in with the driver circuit design to show a final hardware design of the flag.

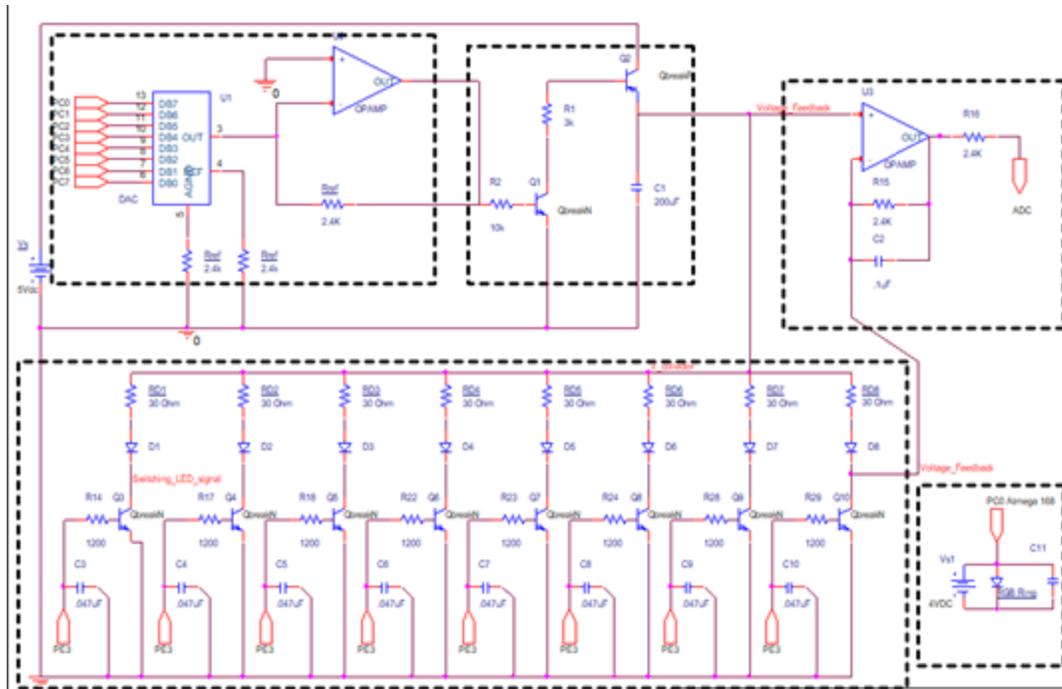


Figure 30 Full Hardware Design

The above figure shows the completed flag circuitry with the 8 IR LED drivers, the dynamic control loop for controlling the capture zone, and the LED ring, to alert the players of the current status of the flag.

On the next page, figure 31 shows the pinout of the ATmega128 and ATmega168 for referencing in figure 30.

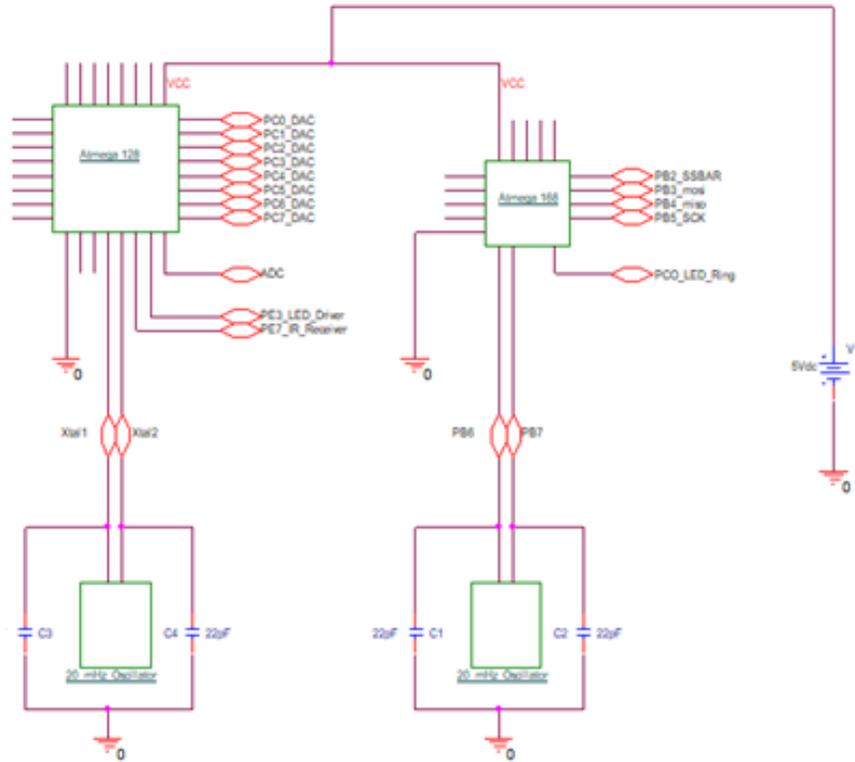


Figure 31 ATmega128 and ATmega168 Pinout Diagram

7. Future Plans

As it stands the following portions of the flag were not completed: SPI on the player side between the ATmega128 and Wi-Fi module, Battery pack circuitry, and dynamic distance control feedback back into the ATmega128. Majority of these incompletions can be repaired and designed with 30 more hours of labor. In regards to future plans of the project in general, this will be discussed below.

Flag

A speaker should be implemented into the design of the flag to act as another method of players within the game to know the status of the flag without having a line of sight of the objective. This would allow for more realistic game parameters and adds to the immersion factor.

Housing for the flag needs to be made to protect all components of the flag. This housing would hold all flag circuitry, batteries, and user interface tools.

Gun

Create a laser tag gun that takes advantage of the communication concepts completed in this project (MQTT and IR). This would allow for full user integration into the flag system and make the games features and objectives completely autonomous.

Implement an RGB LCD display to show all in-game information that a player would have to know such as points, the status of flags, elimination/death ratio, ammo usage, and other objectives. An example of this LCD is shown below in figure 32.

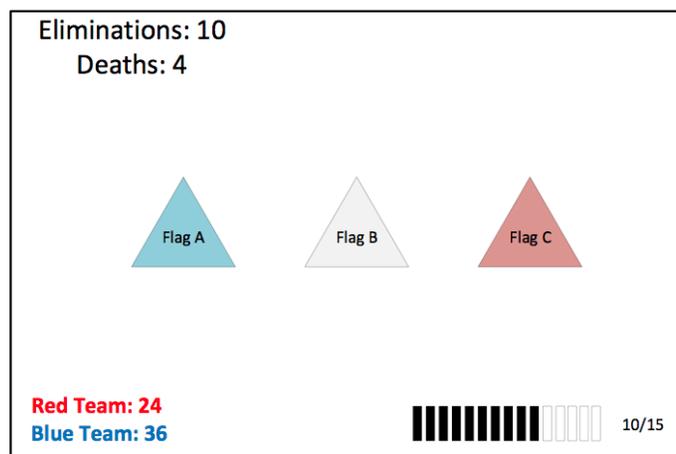


Figure 32 LCD Display

8. Conclusion

The original goals of this project were to have completed all flag hardware and software, and have two working laser tag guns (players). To learn more on each phase of these goals and their design parameters, review “Project Proposal” located on the Autonomous Objective Based Laser Tag webpage. While only the flag was completed for this project, considerable effort and design were put into this project objective. With this senior project being a proof of concept, the year of work showed that this technology is feasible and is possible to both finish and market at a potential later date.

The acknowledgement must be given to Dr. Malinowski, Nick Schmitt, Chris Mattus, Dr. Huggins, and Dr. Gutschlag for advising and giving technical assistance on all portions of the flag design. Thank you.