Final Group Project
Red Light, Green Light

Sean Kennedy
19 August 2022

Squid Game Squad 174

Dimetree Aburto, Vigomar Kim Algador, Shamal Kumar, and Sam Ho

Planning and Team Members Responsibilities:

- Hardware (Shamal Kumar/Dimetree Aburto)
    - Breadboard
    - STM
        - Correct pin implementation
    - LED
        - Correct color to pin combo
    - Motion Sensor
        - Detection Distance
        - Proper Connection
    - Speaker
        - Proper Connection
        - Proper Source/impedance/frequency
- Code
    - Motion Sensor (Vigomar Kim Algador)
        - Sense player movement after Red LED is on
        - Detects foul movement and turns on Flash Code
        - Initiates change of player/loss
    - Buzzer (Sam Ho)
        - Sense input from Red LED to indicate audible loss/player change
        - Turns off after Green Light is Turned on
    - LED's (Shamal Kumar)
        - Proper function when detecting player
        - Proper flach sequence to indicate next player
        - Length of player change Flashing LED reasonable
        - Make a unique LED light up pattern to determine the winner and loser
- Debug
    - Proper registers (Vigomar Kim Algador)
    - state of pins (High or Low) (Sam Ho)
        - Turns on/off
    - Proper Condition Changes (Dimetree Aburto)
        - Jump/Loops have proper function flow
        - Correct Loop Path per FSM Diagram
        - FSM logic to determine the winner
        - FSM logic to change the timer of Red LEDs and Green LEDs

Project Description / Introduction:

The project imitates “Red Light, Green Light” a minigame based off the show “ Squid Games”, but our game will only be intended for a single player at a time. In this lab, we will be using a STM board, motion sensor, green/red LEDs, and a buzzer. There will be a green LED indicating that movement is allowed and a red LED indicating that movement is not allowed. When the red LED is active and its code is executed, the motion sensor activates and detects any movement. If the motion sensor senses movement, the output buzzer will sound off. After a certain time, the green LED will light up indicating the removal of the player was successful and a new play may start.

The code we wrote has event handlers to control what happens at each state. For the GreenLight handler, the green LED is set to light up for 5 seconds; then it turns off and delays for 0.1 second. After that, the code will call the Stop state, and go to the RedLight handler. For the RedLight handler, the Red LED is set to light up and delay for 5 seconds; then it turns off and delays for 0.1 second. After that, the code will call the Move state, and go to the Motion handler. For the Motion handler, our buzzer is assigned to pin 4. The buzzer goes off and delays for 3 seconds; then it turns off and delays for 1 second. This process will repeat twice,then the code will call the GO (Game Over) state, and the game will stop running. Lastly, in the Finish handler, the red LED delay for 5 seconds then it turns off, and the buzzer delays for 1 second and turns off. Which it will call the GO state, and the game stops running as can be seen in **Figure 1**.

```c
typedef enum
{
    Move,
    Stop,
    MD,
    GO
} eSystemState;

 //Green A1, Red A3, Sensor A6, Buzzer A4, Button A5
eSystemState GreenLight(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
    HAL_Delay(5000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_Delay(100);
    return Stop;
}
eSystemState RedLight(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
    HAL_Delay(5000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_Delay(100);
    return Move;
 }
eSystemState Motion(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_Delay(3000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
    HAL_Delay(3000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_Delay(1000);
    return GO;
 }
eSystemState Finish(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
    HAL_Delay(100);
    return GO;
```

```c
itch(eNextState)
{
  case Move:
      eNextState = GreenLight();
      break;
  case Stop:
      eNextState = RedLight();
      if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6))
      {
          eNextState = MD;
      }
      else
      {
          eNextState = Move;
      }
      break;
  case MD:
      eNextState = Motion();
      break;
  case GO:
  eNextState = Finish();
  if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5))
  {
      eNextState = Move;
  }
  else
  {
      eNextState = GO;
  }
  break;
}
```

Figure 1: Relevant Source Code

Block Diagram:

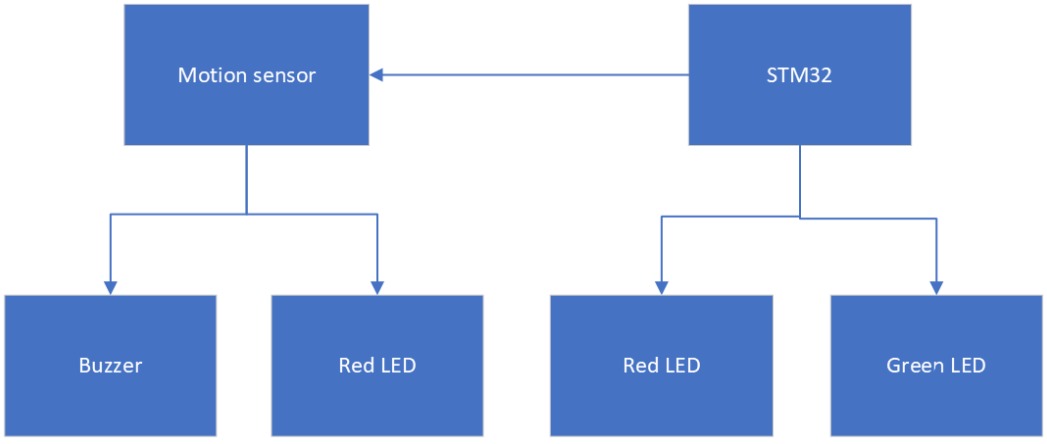

Figure 2: Block Diagram

FSM Diagram/Table

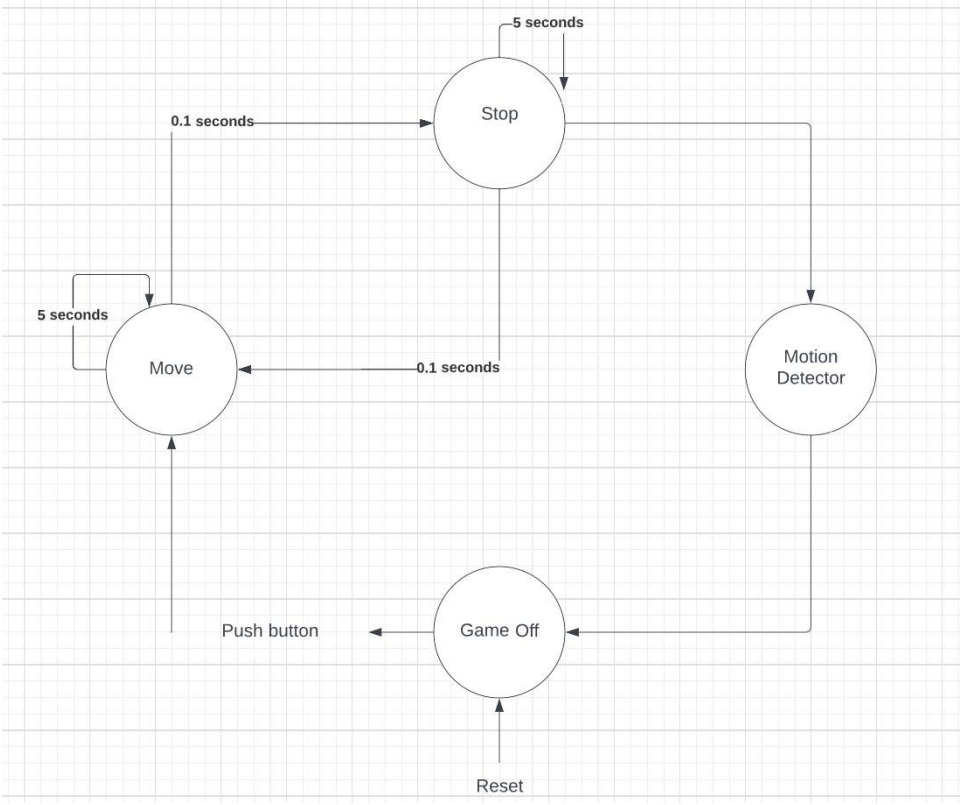

Figure 3 : Finite State Machine

Current State	Input	Next State	Output
Move	Time delay 5 secs	Move	Green_LED = 1 Red_LED = 0
	Time delay 0.1 secs	Stop	
Stop	Time delay 5 secs	Stop	Green_LED = 0 Red_LED = 1
	Time delay 0.1 secs	Motion Detection (MD)	
Motion Detection (MD)	Motion Detector (Pin A6)	Motion Detection (MD)	Buzzer sounds off
		Game Off (GO)	
Game Off (GO)	Push Button (Pin A5)	Game off	Green_LED = 0 Red_LED = 0
		Move	

Figure 4: FSM Table

Figure 3 and **Figure 4** show our FSM diagram and table. The intent is that the Game Off state will be reset.

Background

The following materials are relevant and used for this project:

STM32 microcontroller/Parts

For this project we use STM Nucleo-32 Board (Model Dependent on Person) microcontroller . This microprocessor is able to configure its pins to multiple inputs or outputs. It uses a USB Type-A to power up and has options for the user to use either a 5V or 3.3V output signal for accessory devices. For this lab we will only be using the 3.3V constant output to power our Motion Sensor, Buzzer, and both LED's. Although we were unable to use it, this PLC is capable of printing messages onto a screen using UART. Figure 1A shows one of the many available PLC's that can be used for this project.

- Breadboard
- Red & Green LED: these are *outputs*
- PIR Motion Sensor (AM312): this is an *input*
    - 5 meter detection at 100 degree angle
- 5V Buzzer (ADA1536): this is an *output*
- Tactile push button: this is an *input*
- Jumper cables
- Resistors:
    - 1K Ω Resistor (1x)
    - 470 Ω Resistor (1x)
    - 330 Ω Resistor (2x)

Breadboard Set Up

For the wiring of our breadboard and its components, we start with the STM32 Nucleo microcontroller. Starting at Pin A1 (PA1), it is wired (yellow jumper wire) the green LED in series with a 330 Ω ohm resistor to ground. At Pin A2 (PA3), it is wired (yellow jumper wire) to the red LED in series with a 330 Ω ohm resistor to ground. Next, at Pin A3 (PA4), it is wired (yellow jumper wire) to the positive lead of the buzzer; and the negative lead of the buzzer is grounded. Then, at Pin A4 (PA5), it is wired (yellow jumper wire) to the positive lead of the push button; the negative lead of push button is in series with the 470 Ω resistor, and goes to ground. Lastly, Pin A5 (PA6) is wired to the Vout pin of the motion sensor; on that same node, we put a

1K Ω resistor that then goes to ground. The 3.3V pin is wired to the Vcc pin of the motion sensor, and the motion sensor is grounded at its ground pin.

The following figures, **Figure 4** and **Figure 5**, show our breadboard schematic and actual breadboard set up.

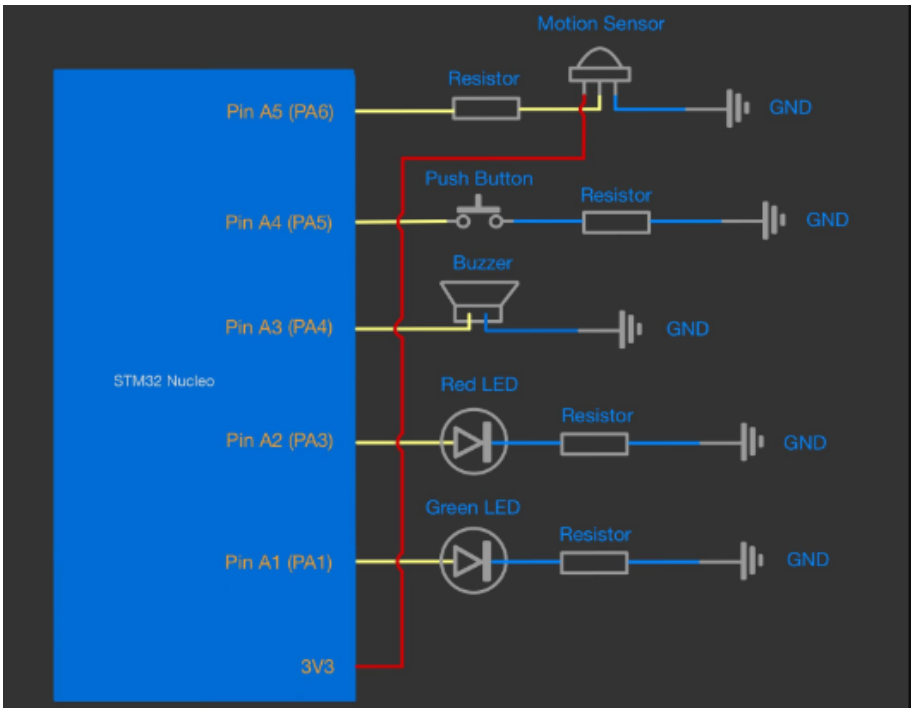

Figure 4: Circuit Schematic

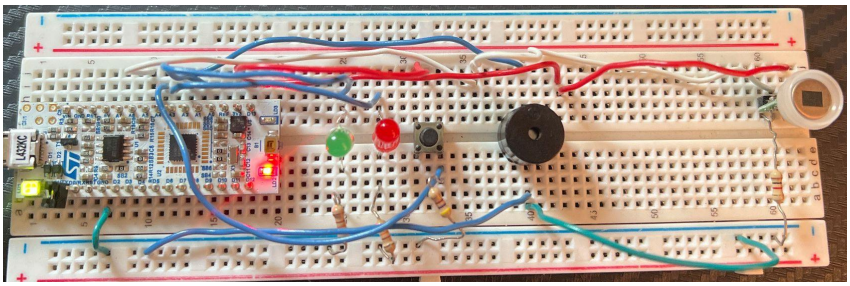

Figure 5: Breadboard Set Up

Results

Test Case 1: **PIR Motion Sensor / Red and Green LED**

Here we are testing to see if the motion sensor adequately captures any movement. For this test we used a Red LED and Green LED. Green LED will be in constant High State when

the Motion Sensor is supplied power, when motion is detected the Green LED will switch to a Low State and a Red LED will turn on. This will give us a visual indication of proper function. The link provided below is a demo of this test case.
Motion Sensor Test Case : https://www.youtube.com/shorts/rD1VaD2fAQE

Test Case 2: **Buzzer**

In this part we are testing our buzzer for the proper communication to the PLC and its Pins. We connect the motion sensor to the positive input of the buzzer and a Green LED. When motion is detected, the buzzer goes off and the Green LED indicates the circuit is closed.
The link provided below is a demo of this test case
Buzzer Test Case : https://www.youtube.com/watch?v=SxclQrjSY0c

Test Case 3: **Debug**

In this part of the program, we go ahead and use the step over function in IDE to determine if the code is running properly. After attempting a few loops of the code, it was deemed worthy of application onto our PLC.

Conclusion:

This project aims to demonstrate a game and the complexity of using PLCs. Along with individual accessories such as a motion sensor, LEDs, and a buzzer. With the knowledge learned from our test cases and the drawn circuit schematic shown in Part 2 of this lab. We all used STMCubeIDE software, selected our nucleo board– some of us have slightly different models– and set up our pin configuration. Before we generate our code, we open a google document to develop our relevant source code. We use Lab 2, polling example and stop light example, as reference and guidance. After a few drafts of code and much discussion, amongst us all, we develop a code we feel confident in. Back in STMCubeIDE, we generate code and write in our source code accordingly. Once we do this, we all run our code. Dimetree had an error occur; to debug this, the team suggested checking his wiring. He fixed his wiring, and the board ran the code for everyone.

Reference

Motion sensor datasheet:
http://www.image.micros.com.pl/_dane_techniczne_auto/cz%20am312.pdf

Buzzer datasheet:
https://www.verical.com/datasheet/adafruit-indicators-and-alerts-1536-5865519.pdf