# Final Project: Guess The Color

Danny Topete
EE/CS 120B

December 9, 2024

## 1 Introduction:

I wrote a guess the color game. The idea originally came from a friend who I saw constantly play CSSbattle.net, I always aspired to be that good at him at guessing the hex values instantly and rebuilding a website like that. So I decided to make a game that would help me get better at it!

In terms of the UI, there is a main menu that gives you a play button. After you press play, a random color is presented to, a prompt to enter the hex code of the presented color. Then a percentage of how close you are to the target value. Along with a timer that records how fast you solve it. As you solve the colors, the shift register turns on one of the three LEDs on the small board.

# 2 Build Upons:

1. **ST7735 Display:** There are two scenes, the start menu, and the playing scene. The start menu's functionality is to show the in the center "PLAY" (which is doing a chroma animation), and, after you press any input, it goes straight into the gamae. This main menu plays a huge role. It serves to keep rolling rand() in the background which almost simulates having a unique srand() seed after every single boot.

   Then, there is the play scene. This is where you have the RGB LED display a

2. **Shift Register:** This was implemented successfully. The only issue is that I believe there are insulation issues, hence why they flicker. When ever I touch the cables, the three LEDs turn off. I notice that when I unplug the VCC, the lights are stable and don't flicker. But this only works while the RED LED is on. I believe this is either poor insulation, or the arduino just doesn't have enough power. It does not take away from the functionality.

3. **IR Sensor / NEC remove:** The IR tasks and sensor works perfectly, I have minimized the amount of times it returns trash values, by setting a higher period. Besides that, sometimes the IR sensor doesn't read if the display is being updated, and again, I believe this is more hardware limitations of the microcontroller.

# 3 User Guide:

- When you begin in the start screen, you will be presented with a simple prompt. To get out of the prompt and play the game, just press any button. You may need to hold down any button until you get the play scene.
- In the play scene, you will be presented with an RGB LED being lit up to a random color and a hex value that you input.

- **Game's Inputs:**
  - **Power Button:** This will roll you a new color if you don't like it
  - **1:** This will iterate your red hex value
  - **4:** This will lower your hex value by one
  - **2:** This will iterate your green hex value
  - **2:** This will lower your green hex value by one
  - **3:** This will iterate your blue hex value
  - **6:** This will lower your blue hex value by one
- You will need to guess the color that the RGB-LED is displaying.
- As you get the closer to the color, the lights in the little board that are connected to the shift register will start outputting values.
  - Example: If you get the value of red correctly, the shift register turn on the Red LED. Along with the display putting a box over the hex value you have correctly guessed.
- After you have correctly guessed the color, you will be greeted with your elapsed time being paused and showing in chroma mode. This means you have won, and pressing the power button will put you into the main menu to start again.

# 4 Hardware:

1. **ST7735 Display:**
2. **RGB LED:** This one was successfully implemented, except for the part where it flickers a bit while the display is updating. I believe this is power issues. But it doesn't take away from the functionality.
3. **Shift Register:** This was implemented successfully.
4. **IR Sensor / NEC remove:**
5. **8 switch DIP:** This has two functionalities, and it works perfectly fine. Pin 7 is my on and off switch. Then my pin 0 is for debugging and to keep rolling new colors. I used this before I came up with a method to always display a pseudo random color.

# 5 Software Libraries Used:

- **avr/io.h**: This library is used for the I/O operations on the AVR microcontroller.
- **stdlib.h**: I used for the rand() function to generate a random color to guess. It keeps the game interesting.
- **avr/interrupt.h**: I used this library to enable the interrupts for the IR sensor.
- **ST77535.h:** Holds the function for sending data/commands, resets the display. Along with the init function for the display. The main functions are Screen(int color) function that fills the screen with that color. Box(short x, short y, short w, short h, short color) that draws a box. Then the bigger brother, fillBox(short x, short y, short w, short h, short color) that colors in the box.

  Then the most important function, Pixel(short x, short y, short color). This function is used by DrawChar(short x, short y, short color, char currVal). To write all my characters. This function alone is 436 lines of code. This function takes my base char of 8, and builds on top of it in a 7-segment like manner.
- **ST77535 Text.h:** This file exclusively holds all 436 linese of the DrawChar(short x, short y, short color, char currVal) function. This function draws a char onto the screen. Using case statements, it knows what character you inputted.
- **helper.h:** SetBit, GetBit were the two important functions out of this file.
- **irAVR.h:** This file is entirely written by the TA. It helps with the funcionality of the IR Remote.
- **spiAVR.h:** I have the default wiring for SCK, MOSI, and SS. This has the SPI Init function, and the SPI Send function.
- **timerISR.h:** This just holds the timerISR function This function is not modified.

# 6 Wiring Diagram:

## Inputs:

**PC0**: IR (Serial Data)

**PC1**: DIP-Switch 7 (0 = low; 1 = High)

## Outputs:

**PB5**: Display - SCK

**PB3**: Display - SDK

**PB2**: Display - CS Pin

**PD7**: Shift Register - SH

**PD6**: RGB - Blue (PWM)

**PD5**: RGB - Green (PWM)

**PD4**: Shift Register - ST

**PD3**: RGB - Red (PWM)

**PD2**: Shift Register - DS

# 7 Tasks Diagram:

1. **RGB TICK:** This is the duty cycle for the RGB LEDs PWM tasks. It handles when it is appropriate to have the LEDs on or off.

2. **DISPLAY TICK:** Helps take care of the two states, the start menu and the active playing menu. When you beat the game, it will strobe rainbow through the time that it took you to solve the color. Along with the start menu, and the strobing rainbow. To strobe the rainbow, I have an array with the colors of the rainbow, and I iterate through them.

3. **IR TICK:** This function is always reading the IR sensor. The power button starts the game from the start screen, and from the play state, it rolls a new color. While it rolls a new color, it resets the time, currVal, and goes to RGB Tick to ask for a new color. Since it grabs the inputs, it updates the currVal when ever it is modified either up or down; along with handling overflow.

4. **RED TICK:** For context, I had issues with digital PWM, and this was my next best option that worked perfectly fine. I was able to get this working within the first week and get this out of the way. The PWM tick function for the green part of the RGB LED.

5. **GREEN TICK:** The PWM tick function for the green part of the RGB LED.

6. **BLUE TICK:** The PWM tick function for the blue part of the RGB LED.

7. **SHIFT Tick:** This runs in a single state to put the value **progress** into the shiftOut() function. The LEDs are only ever turned on when currVal is equal to the target value in the respect hex bits of Red, Green, and Blue.

8. **ELAPSED Tick:** Tracks the elapsed time in a single state. It pauses the elapsed time when you beat the game.

# 8 SynchSM Diagrams: