MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT

EE447 PROJECT FINAL REPORT

Project: Brightness Controller

Student Name: Arda ÜNVER
Syed Muhammad Farrukh AIJAZ

Student ID: 2444081
2417152

Submission Date: 23.12.2023

Table of Contents

1. Introduction

The TM4C123GH6PM microcontroller is utilized in the Brightness Controller Project to construct a system that controls ambient brightness. Through I2C communication and a TSL2561 light sensor, the microcontroller gathers 128 ADC samples, calculates the mean, and converts the unit of the data to Lux. Brightness thresholds are dynamically set via a potentiometer. State transitions are facilitated via a button (SW1) that lets users change thresholds. LEDs that are red, green, and blue show the amount of ambient light. An external high-power LED circuit's PWM control adjusts to variations by raising intensity below the low threshold and lowering it above the high threshold. The report offers information about the functionality and design of the system.

1.1 Components

- TM4C123GH6PM
- TSL2561 – LIGHT SENSOR
- NOKIA 5110 LCD DISPLAY
- 1x POT (10K)
- HIGH-POWER LED
- 2SB772L NPN Transistor
- 2x 120Ω Resistor
- DC Power Source
- Jumper cables

1.2 Pinout of the System

PORTS IN USE: A, B, C, D, F

TSL2561	GND	3V3	SDA	SCL
BOARD	GND	+3.3V	PD0	PD1

LCD	GND	BL	VCC	CLK	DIN	DC	CE	RST
BOARD	GND	VBUS	+3.3V	PA2	PA5	PA6	PA3	PA7

POT	Left Leg	Middle Leg	Right Leg
BOARD	+3.3V	PB4 (ADC)	GND

HIGH-POWER LED CCT	GND	TRANSISTOR (BASE)
BOARD	GND	PC4

SWITCH/LEDS	SW1	RED	BLUE	GREEN
BOARD	PF0	PF1	PF2	PF3

2. Subsystems

2.1 Main Loop Logic

Within the main loop of the Brightness Controller Project, a sequence of critical actions ensures the dynamic and responsive regulation of ambient brightness. The loop initiates by acquiring an ADC sample from the TSL2561 light sensor, capturing real-time data on ambient light intensity. Simultaneously, another ADC sample is obtained from a potentiometer to dynamically set low and high brightness thresholds based on the system's current state.

The system's state is then evaluated using a conditional check. If in state “-1”, indicative of low threshold adjustment, the potentiometer-derived value influences the low threshold; conversely, in state “1”, corresponding to high threshold modification, the potentiometer's reading dynamically updates the high threshold.

Following the threshold adjustments, the detected light value from the sensor is compared with the established low and high thresholds. Based on this comparison, the system selectively activates indicator LEDs. If the detected value falls below the low threshold, a red LED illuminates, signaling insufficient brightness. Conversely, if the value surpasses the high threshold, a blue LED activates, indicating excessive brightness. In the “safe” region, between the low and high thresholds, a green LED is illuminated, denoting the desired illumination level.

Simultaneously, dynamic PWM control is applied to an external high-power LED circuit. If the detected value is below the low threshold, PWM is increased, intensifying the light emitted by the high-power LED to maintain the system within the safe region. Conversely, if the blue LED is activated, PWM is decreased to regulate brightness effectively.

To enhance user visibility into the system's operation, the threshold values, along with the current detected value from the light sensor, are displayed on the Nokia 5110 LCD screen. This visual representation serves as a user-friendly interface, providing real-time feedback on the system's assessment of ambient lighting conditions.

```
while(1){
	// Read potentiometer input (ADC)
	sample = get_pot_data();

	// Poll the SW1 (PF0) to change the state
	check_the_state();

	// Update the thresholds
	if(state == 1){
	   if(sample<high_threshold){
	   low_threshold = sample;
```

```
            delay_100ms();
        }
    }
    else if(state == -1){
        if(sample>low_threshold){
        high_threshold = sample;
            delay_100ms();
        }
    }

    //Get the sensor data from TSL2561 (I2C)
    detected_value = get_lum_data();

    // Update the Duty Cycle
    duty = change_duty_cycle_M0PWM0(detected_value, low_threshold, high_threshold);


    //Setting Indicator LEDs
    if(detected_value>high_threshold){
    GPIOF->DATA|=0x04;
    GPIOF->DATA&=~0x08;
    GPIOF->DATA&=~0x02;
    }
    else if(detected_value<low_threshold){
    GPIOF->DATA|=0x02;
    GPIOF->DATA&=~0x04;
    GPIOF->DATA&=~0x08;
    }
    else{
    GPIOF->DATA|=0x08;
    GPIOF->DATA&=~0x02;
    GPIOF->DATA&=~0x04;
    }
    //Display the data to Nokia LCD
    print_low_data(low_threshold);
    print_high_data(high_threshold);
    print_lux_data(detected_value);
    print_state(state);
}
```

2.1 TSL2561 Light Sensor

2.1.1 I2C Protocol

One common serial communication protocol that makes it simple and effective to communicate between a microcontroller and a variety of peripheral devices is I²C (Inter-Integrated Circuit), which is used in our Brightness Controller Project to interface with the TSL2561 light sensor. With just two bidirectional lines required, SCL (Serial Clock) and SDA (Serial Data), I²C is perfect for devices like the TM4C123GH6PM microcontroller that have limited GPIO availability.

The protocol allows several slave devices, each with a unique address, to be connected to a single bus. When it comes to the TSL2561, the microcontroller functions as the master, starting the SCL line's clock signals and data transmission over its SDA line. This configuration enables light intensity data to be retrieved from the sensor and commands to be sent to it to start measurements.

To establish communication with an I2C slave device, specifically a TSL 2561 sensor, we utilized the I2C3 module on the TM4C123 microcontroller. This involved initializing the I2C3 module and configuring the relevant GPIO pins using the **'I2C3_Init'** function. We then employed the **'I2C3_Write'** function to write data to the TSL 2561 sensor. This write operation was essential to power up the sensor.

To read data from the slave device, we employed the **'I2C3_Read'** function, specifying the slave address, memory address, and the number of bytes to be read. Data was read in four steps: first, we read the low data register of channel 0, followed by its high data register. We repeated this procedure for channel 1's data register.

Here are the functions and their roles:

'I2C3_Init': This function initializes the I2C3 module on the microcontroller by enabling the clock for I2C3 and configuring the necessary GPIO pins on Port D for I2C3. The 'I2C_Error_Check' function is called to ensure that the I2C3 master is not busy before proceeding.

'I2C3_Write': This function writes a series of bytes to a slave device with a specified slave address and memory address. It begins by sending the slave address and memory address to the slave device, followed by sending the data one byte at a time, ending with a STOP condition.

'I2C3_Read': This function reads a series of bytes from a slave device with a given slave address and memory address. It starts by sending the slave address and memory address to the slave device, then switches the bus to read mode by sending a RESTART condition with the slave address and a read bit. It reads the data one byte at a time, sending an ACK after each byte except for the last one, which is followed by a NACK and a STOP condition.

'I2C_Error_Check' : The 'I2C3_Error_Check' routine is used for checking the status of the I2C master to ensure it is not busy before proceeding with further operations. It uses a loop labeled 'WAIT_NOT_BUSY' to repeatedly read the 'I2C3_MCS' register, which contains status information. It checks the busy bit (bit 0) in this register. If the busy bit is set, indicating that the I2C master is still busy, the code continues looping until the I2C master becomes idle.

Once the I2C master is no longer busy, the routine retrieves the I2C error code from the 'I2C3_MCS' register and clears the bits related to error flags (bits 1, 2, and 3). The result, which should be 0 if no errors occurred, is then returned.

'Lux_Calculation': This function calculates lux values from raw data collected by the TSL2561 light sensor. The algorithm is based on constants and formulas provided in the TSL2561 datasheet. To validate the system's accuracy, we tested it in a room with known light levels and compared the readings to a reference lux meter. The system proved to be accurate and reliable in measuring light levels.

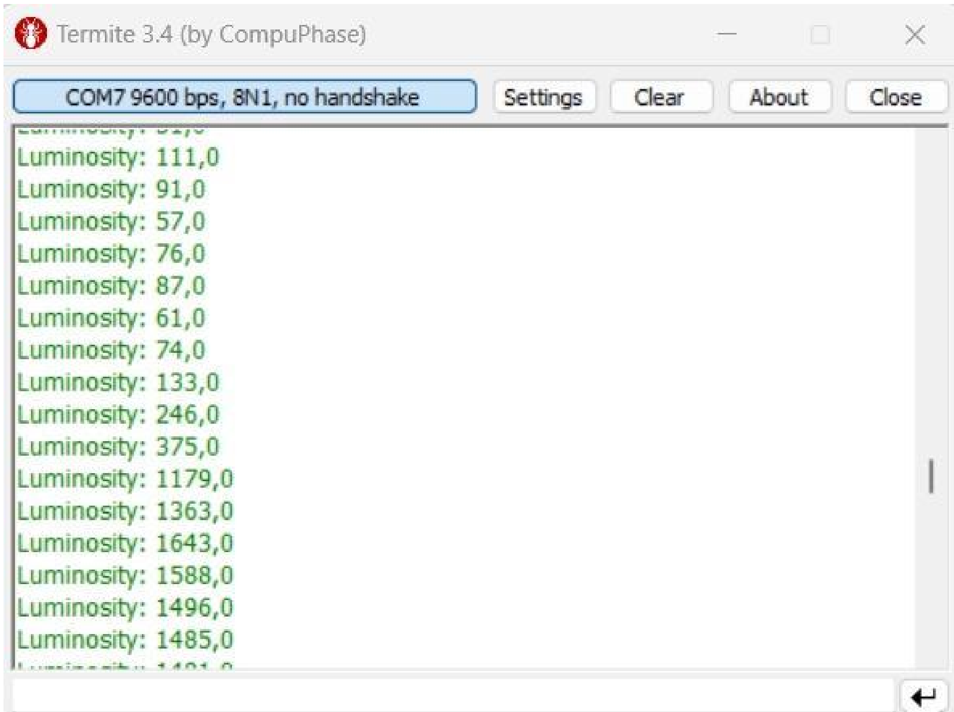

Figure 1. Luminosity Values Calculated from LUX values from the sensor.

2.1.2 Adjusting the received data

We collected 128 samples from both the high and low channels of the sensor by calling functions 'CH0' and 'CH1' and then calculated the average of these values. The resulting mean values were then sent to the **'Lux_Calculation'** function to obtain lux values based on the mean values of channel 0 and channel 1 data registers.

To ensure smooth I2C transactions, we inserted calls to the **'I2C_Error_Check'** function between transactions. This ensured that the I2C3 master was not busy before initiating a new transaction.

Under typical room lighting conditions, the received luminosity values ranged from 10 to 12. When exposed to increased lighting, such as from a cell phone flash, the luminosity reading could reach up to 1500.

Here are the functions and their roles:

'CH0': The core functionality of this code resides in the 'CH0' subroutine. Its objective is to read luminosity data from the TSL2561 sensor. To accomplish this, it sets up the necessary I2C communication parameters, initiates a read request for both low and high data registers of channel 0, collects the received data, and subsequently calculates a 16-bit value by combining the acquired bytes. The final result is stored in register R0, representing the luminosity value.

'CH1': Similar to 'CH0' function, It initializes the I2C communication parameters, initiates a read request for both low and high data registers of channel 1, retrieves the data, and calculates a 16-bit value by combining the received bytes. The resulting luminosity value is stored in register R0.

2.2 Potentiometer and Thresholds

The Analog-to-Digital Converter (ADC) plays a crucial role in obtaining input from a potentiometer, allowing for dynamic adjustment of lighting thresholds. The adc_init function initializes the ADC module of the TM4C123GH6PM microcontroller. This setup involves enabling the ADC0 module and configuring the appropriate GPIO port (in this case, Port B) for analog input. The pin PB4 is set to function as an ADC input by enabling its alternate function (AFSEL), setting it as an input (DIR), disabling its digital function (DEN), and enabling its analog function (AMSEL). In the ADC configuration, Sequencer 3 is used with software triggering (EMUX), and a single sample is taken (SSCTL3). The selected input channel is AIN10 (SSMUX3), with the configured sampling rate set through the ADC0->PC register.

The **get_pot_data** function retrieves the ADC data, representing the potentiometer position. This function triggers a new ADC conversion, waits for the conversion to complete (checked using ADC0->RIS), and then reads the converted value from the ADC0->SSFIFO3 register. The raw ADC value is scaled to a range suitable for your application (in this case, scaled to a range of 0 to 1500), providing a meaningful representation of the potentiometer's position.

The **check_the_state** function is responsible for toggling the system's state based on the user's interaction with the switch (SW1, connected to PF0). Initially set to -1, the state alternates between 1 and -1 with each press of the switch. This toggling is accomplished by detecting a low signal on PF0 and debouncing the switch press with a delay (implemented via **delay_100ms**) to prevent erroneous readings due to the mechanical nature of the switch.

In the main loop, the potentiometer data is continuously read, and the state is checked to determine whether to update the low or high threshold. When the state is 1, the low threshold can be modified, provided the new value is lower than the high threshold. Conversely, when the state is -1, the high threshold is adjustable, ensuring it remains higher than the low threshold. This mechanism of threshold adjustment enables dynamic control over the lighting conditions.

Finally, the system uses the obtained thresholds and the detected value from the light sensor (TSL2561) to control the indicator LEDs, as it can be seen in *Figure 2*. If the detected light level falls below the low threshold, a red LED is activated (PF1). If it exceeds the high threshold, a blue LED is illuminated (PF2). For light levels within the thresholds, a green LED (PF3) is lit, indicating the desired or safe lighting range. This visual feedback is essential for monitoring and maintaining the lighting conditions in your project's environment.

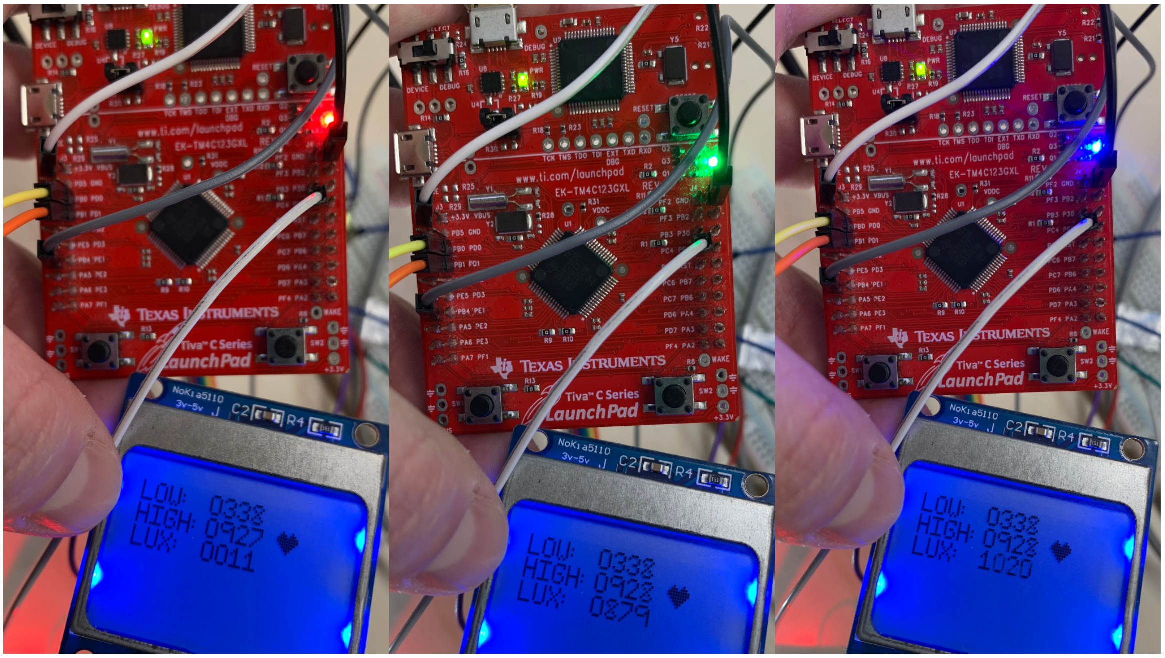

Figure 2. On board LED's change of state when LUX value changes.

2.3 High-Power LED Circuit

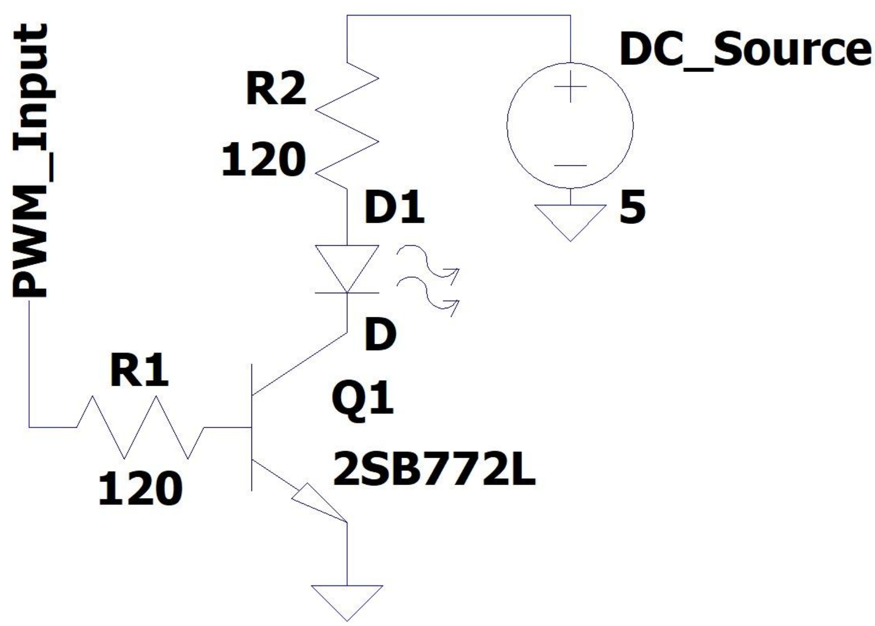

Figure 3. Spice Model of the High-Power LED Circuit.

The circuit is designed to be controlled by a Pulse Width Modulation (PWM) input signal. PWM is a technique used to encode a message into a pulsing signal. In the circuit, shown in *Figure 3*, the PWM input dictates the operation of the transistor switch. Two resistors, R1 and R2, both marked with a value of 120 ohms, are included in the circuit. R1 is connected to the base of the transistor and serves to limit the current entering the base, ensuring that it stays within safe operating limits for the transistor. R2 functions as a pull-up resistor, connected to the collector side of the diode D1, maintaining a high state when the diode is not conducting. During the 'on' phase of the PWM, current flows through R1 into the base of Q1, turning it on. This allows current from the DC source to flow through the LED connected to the collector of Q1. During the 'off' phase, no base current flows, and Q1 turns off, stopping the current flow to the LED. The average voltage and power delivered to the LED is controlled by the duty cycle of the PWM signal. If the PWM signal has a high-duty cycle (more time on than off), more power is delivered to the LED. Conversely, a low-duty cycle results in less power to the LED.

Pulse Width Modulation (PWM) is employed to regulate the intensity of a High-Power LED circuit based on the luminosity values detected by the TSL2561 light sensor. The **pwm_init_M0PWM0** function initializes the PWM module (PWM0) of the TM4C123GH6PM microcontroller, configuring the necessary system clocks for the PWM module and GPIO port C. Port C, pin 4 (PC4) is set up as the output pin for the PWM signal, with appropriate settings in the GPIO configuration registers (such as AFSEL, PCTL, and DEN) to define its alternate function as a PWM output. In this setup, PWM Generator 3 is configured in down-count mode, with a specific load value (5000) to achieve a 1 kHz frequency and an initial duty cycle of 50% (CMPA set to 2500-1). The PWM signal is then enabled on the respective channel, ready to control the High-Power LED's brightness.

The **change_duty_cycle_M0PWM0** function dynamically adjusts the PWM duty cycle based on the real-time luminosity readings. The function takes sensor data, along with predefined low and high threshold values (LT and HT), to determine the necessary adjustment. If the sensor data exceeds the high threshold, indicating excessive brightness, the duty cycle of the PWM signal is decreased, thereby reducing the intensity of the High-Power LED (signified as the 'blue region'). Conversely, if the sensor data falls below the low threshold, suggesting insufficient brightness, the duty cycle is increased to enhance the LED's brightness ('red region'). In scenarios where the sensor data is within the thresholds ('green region'), the duty cycle is reset to a neutral value (50%). The duty cycle adjustments are constrained within certain limits (between 100 and 4800 in this case) to prevent overdriving or complete shutdown of the LED. This responsive adjustment ensures that the lighting conditions are constantly regulated to maintain an optimal luminosity level as detected by the sensor, with the direct feedback loop between the sensor and the High-Power LED enabling real-time adaptability of the lighting environment.

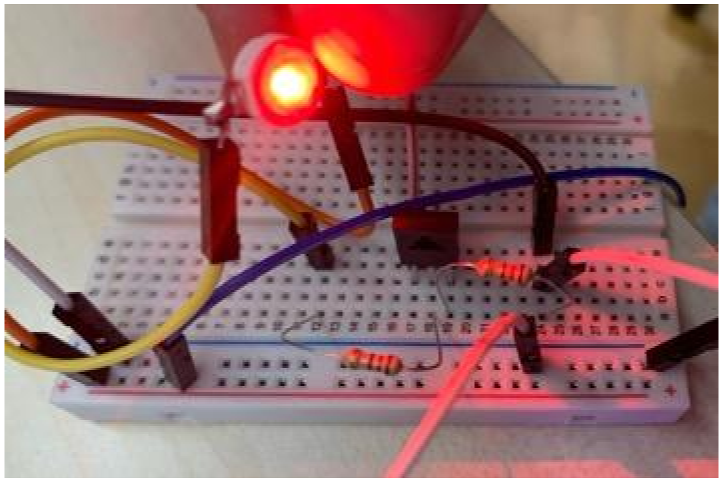

Figure 4. High-Power LED Circuit.

2.4 Nokia 5110 LCD Display

2.4.1 SPI Protocol

Employing the **TM4C123GH6PM** microcontroller for our brightness controller project, the Nokia 5110 LCD display is interfaced with the help of the Serial Peripheral Interface (SPI) communication protocol, namely the **SSI0** module. Synchronous serial communication (SPI) is a protocol that allows microcontroller-based systems to interface with a wide range of peripherals, including LCDs, memory sticks, and sensors. It operates on a master-slave architecture, where the microcontroller typically acts as the master, controlling the communication flow. Four main communication lines are used by the SPI protocol: **SS** (Slave Select), **MISO** (Master-In-Slave-Out), **MOSI** (Master-Out-Slave-In), and **SCLK** (Serial Clock).

Data is sent from the microcontroller to the LCD via the MOSI line. Although necessary for SPI, the MISO line is not utilized in this particular situation since the LCD does not transmit data back to the microcontroller. Finally, the Nokia 5110 LCD is selected as the active device on the SPI bus via the SS line, which may be manually adjusted via GPIO.

The SPI protocol enables it to be easier to transmit data efficiently for the Nokia 5110 LCD. With the SSI0 module of the TM4C123GH6PM set up for SPI communication, commands and information are sent to the LCD to manage its operations, including displaying numbers, text, and images, adjusting contrast, and determining the position of the cursor. These tasks are completed quickly considering the SPI's high-speed serial data transfer capability, which makes it the perfect option for real-time display applications. Accurate control over the timing of data transfer is ensured by the synchronous nature of SPI and its clock line, which is essential for the LCD to operate correctly.

This precise timing control is extremely beneficial for updating the display with data that changes rapidly, such as the project's real-time light intensity readings. Since the Nokia 5110 LCD functions as a user-friendly interface that offers real-time feedback and improves the project's interactivity, the usage of SPI with the LCD significantly enhances to the project's total capability.

The **display_init** function is critical for preparing the LCD to receive data. It begins by controlling the reset (RST) pin connected to PA7: the RST pin is first set low and then high after a delay, ensuring the LCD is properly reset. The Data/Command (D/C) pin, connected to PA6, is then set low to indicate that subsequent transmissions will be commands, not data. The function proceeds to send a series of initialization commands to the LCD through the SSI Data Register (SSI0->DR), with each command configuring specific aspects of the LCD, such as function set, contrast (Vop), temperature coefficient, LCD bias, and the standard instruction set. The display is then switched to normal mode, and the cursor is positioned at the beginning (X=0, Y=0).

The **SSI0_init** function is essential for setting up the SSI0 module, which facilitates communication with the LCD. This setup involves enabling the SSI0 module and the required GPIO ports (A and B). GPIOA is configured to support SSI communication, with PA2, PA3, PA4, and PA5 set for their alternate function as SSI0 pins. The SSI0 module is configured in master mode, with a specific prescale value and serial clock rate to ensure correct data transmission speed and format. The SSI data format is set to 8-bit, and the clock polarity and phase are cleared, which is necessary for proper synchronization with the LCD. After these configurations, a check is performed to ensure that the SSI is not busy before proceeding with further operations.

Together, these functions establish a robust communication interface with the Nokia 5110 LCD, allowing for the display of information such as sensor readings, system status, or other relevant data. The initialization sequence ensures the LCD is correctly configured and ready to display data received via the SSI0 module. This display functionality is a crucial aspect of your project, providing a user-friendly interface for real-time monitoring and feedback.

2.4.2 Writing Data on Display

By default, the LCD works in horizontal mode, with five columns of space allocated to each character—number or letter. The "BSY" (Busy) bit in the SSI status register must first be checked before writing data onto the display. This step is essential to prevent data transmission conflicts and guarantee that previously provided data has been correctly received by ensuring that fresh data is only loaded into the SSI Data Register (SSI0_DR) when the module is not busy. The GPIOA port is used, with the Data/Command (D/C) pin connected to PA6, to initialize and control the display.

The state of this D/C pin indicates the type of data being delivered; a low setting denotes a command transmission and a high setting denotes data transfer. For example, in "DATA" mode, a series of bytes (0x7C, 0x82, 0x92, 0x92, 0x66) is communicated consecutively in order to display the letter "G" on the LCD. These bytes are transmitted as data after the required command to establish the cursor position on the display. Each byte represents a column of the character in a 5x8 pixel format.

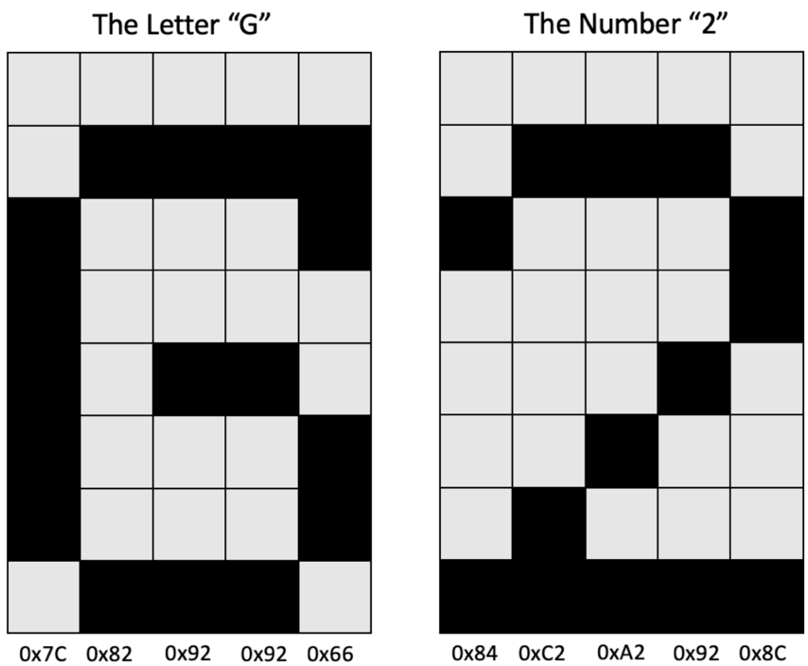

Figure 5. Pixel Encoding shown for different characters and numbers.

2.4.3 Display Functions

Initialize Display. void display_init(void);	Configure SPI. void SSI0_init(void);
100ms Delay. void delay_100ms(void);	Write a byte to the data register. void write_DR(uint8_t data);
Set mode of operation, DATA or COMMAND. void set_mode(enum operation MODE);	Clear the screen. void clear_screen(void);
Set the cursor on a specific location on the display. void set_cursor(uint8_t x_cursor, uint8_t y_cursor);	Print a number at a significant location on the display. void print_number(int num, uint8_t x_cursor, uint8_t y_cursor);
Write a single digit. void print_digit(int num);	Write "LOW:" on the display. void string_low();
Write "HIGH:" on the display. void string_high();	Write "LUX:" on the display. void string_lux();
Write the labels on the screen. void configure_screen();	Print low_threshold data on the display. void print_low_data(int data);

Print high_threshold data on the display. void print_high_data(int data);	Print luminosity (or detected_value) on the display. void print_lux_data(int data);
Print the indicator symbol. void print_heart();	Clear the indicator symbol. void print_blank();
Print a symbol on the display to indicate which threshold is to be changed. void print_state();	

Figure 6. Programmed Display shown in Nokia 5110 LCD

3. Test Results

Link to the DEMO:
https://www.youtube.com/watch?v=9stsWf7276U

4. References

1. TM4C123GH6PM Datasheet
https://www.ti.com/product/TM4C123GH6PM

2. TSL2561 Light Sensor Datasheet
https://cdn-shop.adafruit.com/datasheets/TSL2561.pdf

3. Nokia 5110 LCD Display Datasheet
https://datasheetspdf.com/pdf/1418219/Nokia/5110/1