ECEN-435
Embedded Microcontroller Design

Final Report

Osama Atta

Fall 2022

Department of Electrical
&
Computer Engineering
University of Nebraska-Lincoln
(Omaha Campus)

Table of Contents

1 Summary

This project entailed the design and development of a Printed Circuit Board (PCB) that utilizes serial communication protocols to transmit and receive data between its various surface mounted components. The components integrated onto the embedded system are a STM32 microcontroller, proximity sensor, camera, SRAM, USB to UART, GPS module, and a WIFI module. After determining the correct protocol for each of the components and creating a block diagram to display that information, EasyEDA was used to create the schematic. EasyEDA was used as it was straight forward to use and had all the tools required to complete this project. After designing the schematic and PCB in EasyEDA, the board was ordered using JLCPCB as they proved to be the cheapest option and had a quick delivery time. After receiving the board, the continuity of the traces was checked, and components were soldered on using a reflow oven. After soldering all the components, the code was written in embedded C using STM32CubeIDE and flashed onto the STM32 microcontroller on the board. The code was designed to interface the components listed earlier with the STM32 and will be discussed further throughout the report. This report will discuss the block diagram generation, schematic design, PCB design and manufacture, component placement and soldering, code creation, and testing.

2 Objective

The objective of this lab and project was to design and develop a PCB that uses serial communication protocols to interface with various components. Another objective of this project was to write code in embedded C to interface the various components with the STM32 microcontroller. The project was deemed successful when the STM32 microcontroller was able to receive and transmit data to the proximity sensor, camera, SRAM, USB to UART, GPS module, and WIFI module. The proximity sensor had to be able to transmit data relating to the distance of objects to the sensor. The camera had to be able to transmit a picture to the STM32 that is then sent and displayed to the LCD on the board designed in ECEN 433. The SRAM had to store data transmitted from the STM32. The USB to UART had to be able to transmit data from the STM32 to a serial monitor and receive data from the serial monitor to the STM32. The GPS module had to transmit data to the STM32 that displayed the coordinates in various NMEA sentence formats. The WIFI module had to act as the bridge between the board designed in this lab and the board designed in ECEN 433.

3 Introduction

An understanding of the inter-integrated circuit (I2C), Serial Peripheral Interface (SPI), and the Universal asynchronous receiver-transmitter (UART) serial communication protocols are required to interface with all the aforementioned components on the PCB. I2C is a half-duplex, synchronous serial protocol that possess only two lines: a serial data line (SDA) and a serial clock line (SCL). SPI is a full duplex, synchronous serial protocol that possesses four lines: Master In Slave Out (MISO), Master Out Slave In (MOSI), Slave Select (SS), and Clock (SCLK). Finally, UART is a full-duplex, asynchronous serial protocol that possess only two lines as well: Transmitter (Tx), and Receiver (Rx). How each of these serial protocols were used in this project will be discussed throughout the report.

Another critical prerequisite to completing this project was understanding reflow soldering and the procedure behind using a reflow oven. Reflow soldering is when controlled heat is applied to components that melts the solder paste and connects the component to its contact pad on the PCB. The soldering oven is used to supply that controlled heat. The datasheets of the components supply the reflow profile which details the heat requirements. That information is required when programming the reflow oven so that the components are not damaged, and the appropriate heat is used to melt the solder paste. The GPS board served as an introduction on how to use a reflow oven.

STM32CubeIDE was the integrated development environment used to develop and deploy the embedded C code onto the STM32 microcontroller. This IDE supplied hardware abstraction layer (HAL) libraries that contain functions that make utilizing the aforementioned serial communication protocols simple. The IDE also possessed debugging features that were useful when facing issues in the code. The software implementation will be discussed later on in this report.

4 Hardware Discussion

The microcontroller used in this project is the STM32L031K6T7. Two external circuits were designed for the controller: one for the reset pin, and one for the BOOT0 pin. When BOOT0 is low and the controller is reset, the controller runs the code in the memory. Pins A2 and A3 of the controller were used as TX and RX respectively to interface with the ESP WIFI module via the UART protocol. Pins A5, A6, and A7 were used as the clock, MISO, and MOSI respectively for the SPI serial protocol. Pins A9 and A10 were used as the TX and RX respectively to interface the controller with the USB to UART component via the UART serial protocol. Pin A11 of the controller was used as the slave select for the camera and pin A12 was used as the slave select for the SRAM. Pins A13 and A14 were used as SYS_SWDIO and SYS_SWCLK respectively which are used to program the STM32 using an ST-Link. Pin B5 was used as an interrupt pin for the proximity sensor. Pins B6 and B7 were used as the SCL and SDA lines of the I2C protocol respectively. A 4.7k ohm pull up resistor was connected to the SDA line and one was connected to the SCL line. The following components were connected to the STM32 controller:

1. Proximity Sensor: VCNL3040
2. Camera: Arducam 2 MP Plus
3. SRAM: S62WVS2568GBLL-45NLI-TR
4. USB to UART: FT231XS-R
5. GPS Module: MAX-8C
6. WIFI Module: ESP-WROOM-02

The proximity sensor was connected to the microcontroller via I2C and was powered using 3V. A 10k ohm pull up resistor was used for the interrupt pin as instructed by the datasheet. The Camera was connected to the STM32 using both I2C and SPI and powered using 5V. The SRAM was connected to the controller using SPI and was powered using 3V. The HOLD pin of the SRAM was pulled high as instructed by the datasheet. The USB to UART chip was connected to pins A9 and A10 of the controller. The USB to UART chip was connected to the Micro USB connector using the bus powered configuration given in the datasheet. The CBUS 1 pin of the USB to UART chip was connected to an LED which lit up when there was data on the TX line and CBUS 2 was connected to an LED which lit up when there was data on the RX line. The GPS module was connected to the I2C bus and powered using 5V. The WIFI module was connected to pins A2 and A3 of the microcontroller. A jumper was placed on the TX line since the TX line cannot be driven by the controller while programming. A button that pulls down the reset and IO0 pins was added so that the ESP can be put into programming mode. To power the board, a DC power jack was placed that supplied the board with a 5V source. To get that 3V that many of the components use, a LDO voltage regulator was used. A slide switch was also added to power the board on and off.

After all the circuitry detailed above was put into a schematic in EasyEDA, the PCB was generated. The top layer was used for the horizontal connections and the bottom layer was used for the vertical connections. The track width was set to 0.25mm and the clearance was set to 0.15mm. The only exception to this design rule was for the 3V lines which had a track width of 0.64mm and a clearance of 0.15mm. Headers were used for all the pins so that logic analyzers and multimeters could be easily connected to diagnose any issues with the circuit. After the design was finalized and approved by Mr. Boeding, the board was ordered via JLCPCB and arrived within a week. The final board was small at 112.014mm*74.76mm.

After the continuity of all the traces was checked, the components were soldered on with solder paste and the reflow oven. A stencil was ordered with the PCB from JLCPCB to make applying the solder pasted easy. After applying the solder paste, the components were placed according to the design and then the PCB was put into the reflow oven. The reflow oven was set to program 0 as it seemed to be within the limits of all the components. After the reflow oven was done, the board was checked for solder bridges. Continuity between pins that were neighbors was checked to make sure that there were no hard-to-spot solder bridges.

4.1 Block Diagram

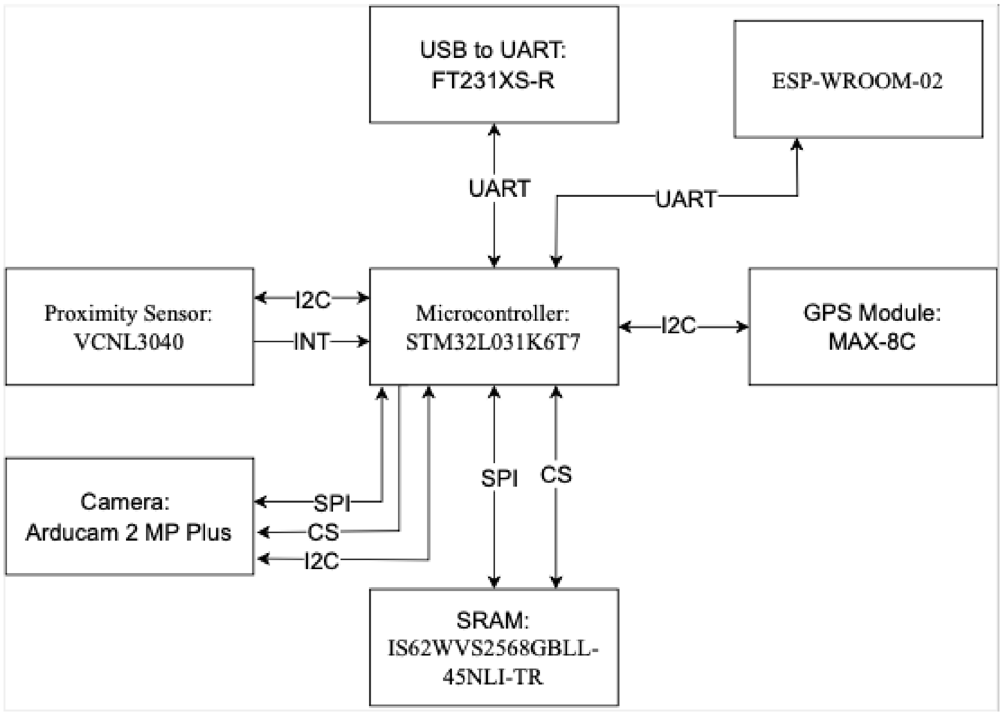

Figure 1: Block diagram of the embedded system showing which protocols are being used.

5 Software Discussion

5.1 Function Blocks

The code for this project was written using embedded C using STM32CubeIDE. After starting a project and designating what pins will be used for what serial protocols in the IOC file, the IDE generates some foundation code that initializes the pins. The IDE also provides HAL functions that can be used to transmit and receive data on the various serial communication lines. The main function of the code initializes the proximity sensor, RAM, and camera. The USB to UART is used to display a shell on a serial monitor and is the main point of user control over the board. The baud rate for the USB to UART was set to 115200. After initialization of the components the main function enters an infinite loop that prompts the user to select one of four options via the serial monitor corresponding to the various components on the board which will be discussed in this section.

5.1.1 main

The main starts off by setting the slave selects for both the SRAM and camera to indicate they are not being used yet. The first component to be initialized is the proximity sensor. This is done by the sensor_init() function. This function sends 0x00 to the PS_CONF3 register, 0x03 to the PS_MS register, 0x40 to the PS_CONF1 register, and 0x08 to the PS_CONF2 register. These settings enable to sensor to send 16 bit data and set the LED current selection to 120mA. The device address of the sensor is 0x60. Next the RAM is initialized. This is done using the ram_init() function. The function simply sends 0x00 to the write mode register (located at 0x01) which sets the ram to byte mode. Then the camera is initialized by the cam_config() function. This function initializes the camera via I2C with 193 values that were supplied by Mr. Boeding. The device address of the camera is 0x30. The main function then enters an infinite loop where the user is prompted to select between different options that allows for the use of the various components.

5.1.2 CAM

The CAM option in the shell menu used the camera to take a picture and the ESP to send it over to the LCD on the board from ECEN 433. Firstly, we wait for the ESP to send the letter 'R' to the controller which indicates that the ESP is ready for data. After the ESP is confirmed to be ready, the cam_pic() function is called. First 0x20 is sent to the 0x04 register via SPI to reset the FIFO read pointer of the camera. Then 0x10 is sent to the 0x04 register to reset the FIFO write pointer. Then 0x02 is sent to the 0x04 register to start the capture. Next, the function enters a loop that constantly checks the 0x41 register for the camera done FIFO write flag to check if the picture is done being taken. After the picture is done being taken and the loop is exited, 0x3C is sent to the camera via SPI to indicate that we are ready to start the burst FIFO read operation to get our picture. Before reading the data, "IMG" is sent to the ESP via UART to indicate that we are ready to send image data via WIFI. After that is done, the function starts to read 320*240 pixels. Each pixel is 16 bits that is split into a high and low byte. While reading the 320*240 pixel image, the images that are within the center 120*160 pixels of the image are sent to the ESP via UART. The Baud rate for the ESP was set to 115200. After the entire picture is read, the pin SS pin is set and we return from the function.

5.1.3 GPS

The GPS option in the shell menu simply receives data from the GPS module and prints it to the shell. This is done via the getGpsData() function that first transmits 0xFF to the GPS module via I2C to indicate that the controller is ready to receive data. The device address for the GPS is 0x42. There is then a delay of 40ms that is done using the HAL_Delay() function. The controller then receives 256 bytes worth of data from the GPS module and stores them into an array. The first 162 bytes from the array are then printed out to the serial monitor.

5.1.4 SEN

The SEN function in the shell enters an infinite loop where the getSensorValue() function is called. This function simply takes a pointer as a parameter. This pointer points to the location where we want the value from the sensor to be stored to. The HAL_I2C_Mem_Read() function is used to read the data from the PS_Data_L and PS_Data_M registers. The function then takes the data read from these two registers and then stores a hex value into the location pointed to by the pointer parameter. A higher value in PS_Data_L corresponded to an object being closet to the proximity sensor. The range of values returned by the getSensorValue() function ranged from 0x31 to 0x39 with 0x31 meaning an object was very close to the sensor.

5.1.5 RAM

The RAM option is very simple and is used to demonstrate that the ram is functional. The RAM option prompts the user to enter a hex value that the user would like to use to check the functionality of the RAM with. The value that is entered by the user on the serial monitor is then written to a preselected address in RAM using the ram_write_split() function and then read back using the ram_read_split() function. If the value read back from RAM is not the same as the value written, then an error message is printed. Else the program will just move on.

5.2 Sequential Diagram for Program

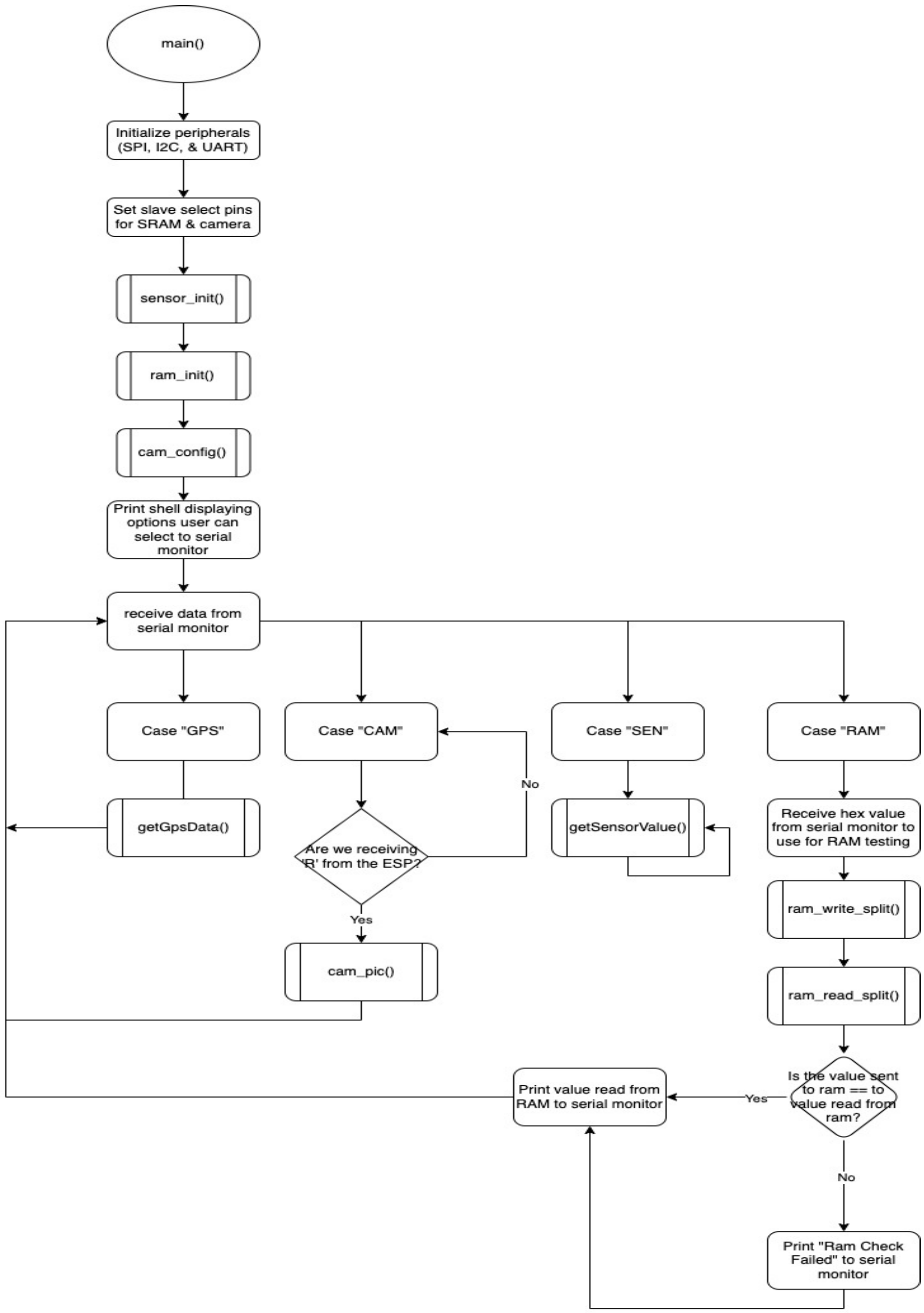

Figure 2: Flowchart for main()

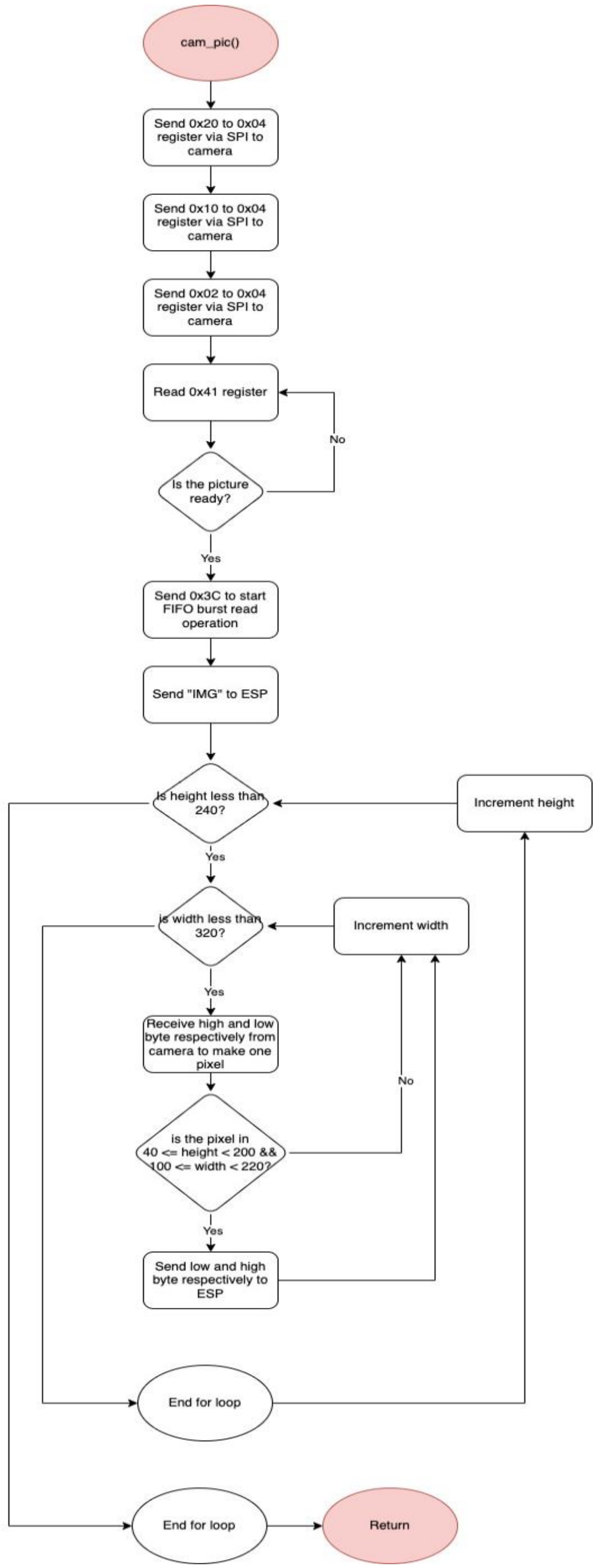

Figure 3: Flowchart for cam_pic()

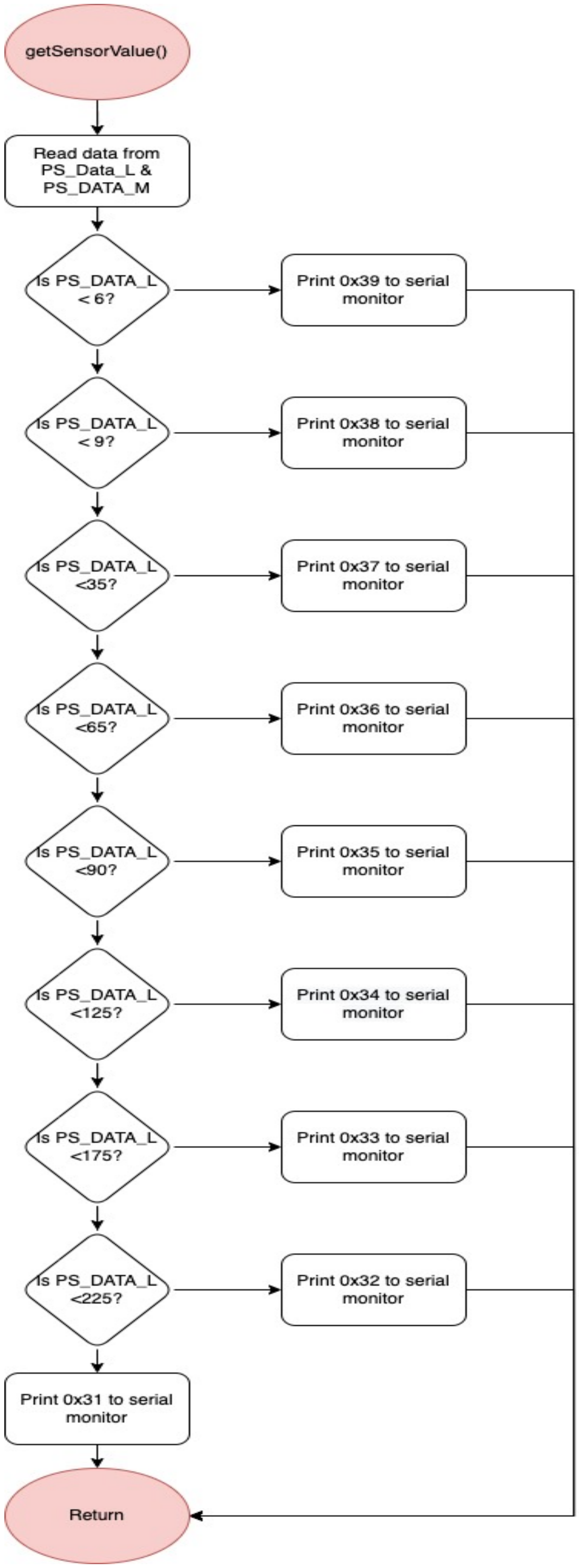

Figure 4: Flowchart for getSensorValue()

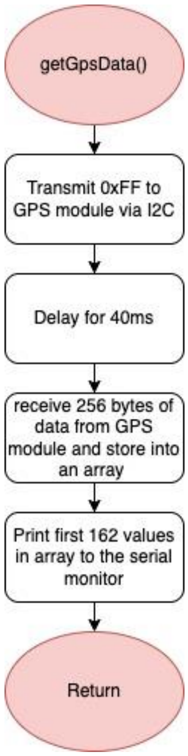

Figure 5: Flowchart for getGpsData()

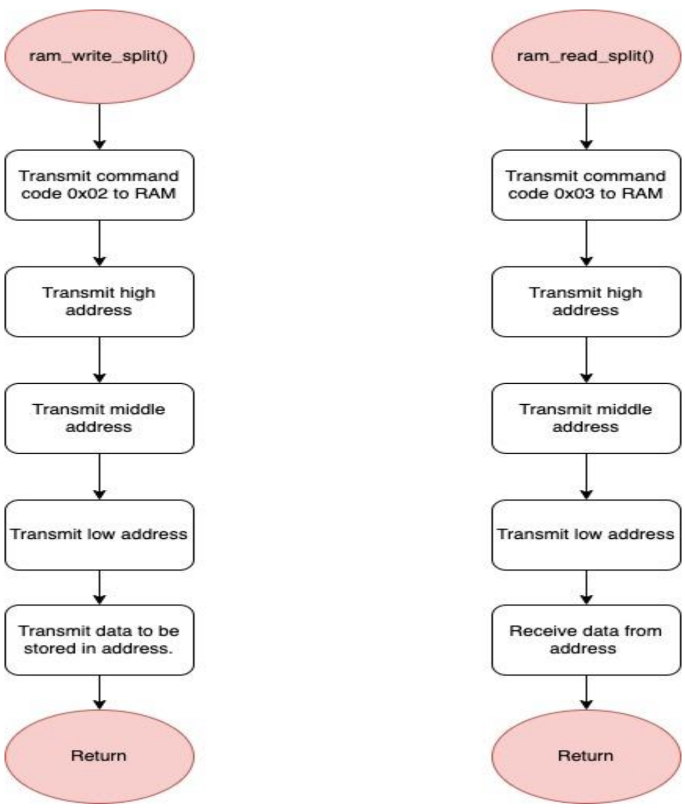

Figure 6: Flowchart for ram_write_split() (left) and ram_read_split() (right)

5.3 UML

main
SRAM_WRMR: const uint8_t
SRAM_SEQ: const uint8_t
configs: uint8_t [ ] [2]
charToHex(val: char): unsigned int
printToMonitor(data: uint8_t*, size: uint8_t): HAL_StatusTypeDef
sensor_init(): HAL_StatusTypeDef
getSensorValue(data: uint8_t*): HAL_StatusTypeDef
getGpsData(): HAL_StatusTypeDef
ram_init(): void
ram_write_split(high: uint8_t, med: uint8_t, low: uint8_t, value: uint8_t): void
ram_read_split(high: uint8_t, med: uint8_t, low: uint8_t, data: uint8_t*): uint8_t
cam_config(): HAL_StatusTypeDef
esp_tx(data: uint8_t): void
esp_rx(): uint8_t
cam_pic(void): void

6 Problem Discussion

The project went smoothly for the most part. The only issue faced during this project related to the camera. I was aware that the camera would be the most complex to setup, so I started by making sure that all my other components were functioning first. After getting all my other components to work, I moved on to the camera. This was a good idea because it allowed me to make sure that all the lines were functioning and that there was no issue transmitting data using SPI and I2C which were both used by the camera. After initializing the camera with the array supplied by Mr. Boeding over I2C and making sure that the camera was returning ACKs I was sure that the camera was being initialized. I tried multiple different ways of reading the FIFO register of the camera and all the pictures I was getting were random colors in random formations. I tried using Arducam software to make sure that the camera was functional however, the software would not detect my camera. I then borrowed a camera from Jonathan Boissy and instead of connecting it to 5V I connected it to 3V and that gave me a clear picture. What I believe happened with my own camera was having it connected to the 5V source for long periods of time was making it malfunction as it was heating up quite drastically. I would avoid this issue in the future by making sure that all my components were at a safe operating temperature.

7 Conclusion

This project saw the design and development of a PCB that utilizes I2C, SPI and UART serial communication protocols to transmit and receive data between its various surface mounted components. The components utilized in this embedded system are: a STM32 microcontroller, a camera, USB to UART, a proximity sensor, a GPS module, SRAM, and a WIFI module. The schematic and PCB design were done using EasyEDA and the PCB was manufactured and delivered by JLCPCB. After making sure of the continuity of all the lines, the components were soldered on using a reflow oven. After soldering the components code was written to interface all the components with the STM32 microcontroller. The project was deemed successful when all the components were able to interface with the STM32 and were able to perform their functions. The camera was programmed to send a data over WIFI to the board designed in ECEN 433 to display a picture on the LCD. The USB to UART was used as the main point of user control over the embedded system by allowing the user to access a shell that allowed for the user to select between some commands that utilized the various components on the board. This project was a great way of employing all that we have learned about serial communication in a practical form that would be used in industry. It was also a great way to explore what modern embedded system design looks like with the use of the STM32 and the STM32CubeIDE.

8 Appendix

8.1 Schematic Design

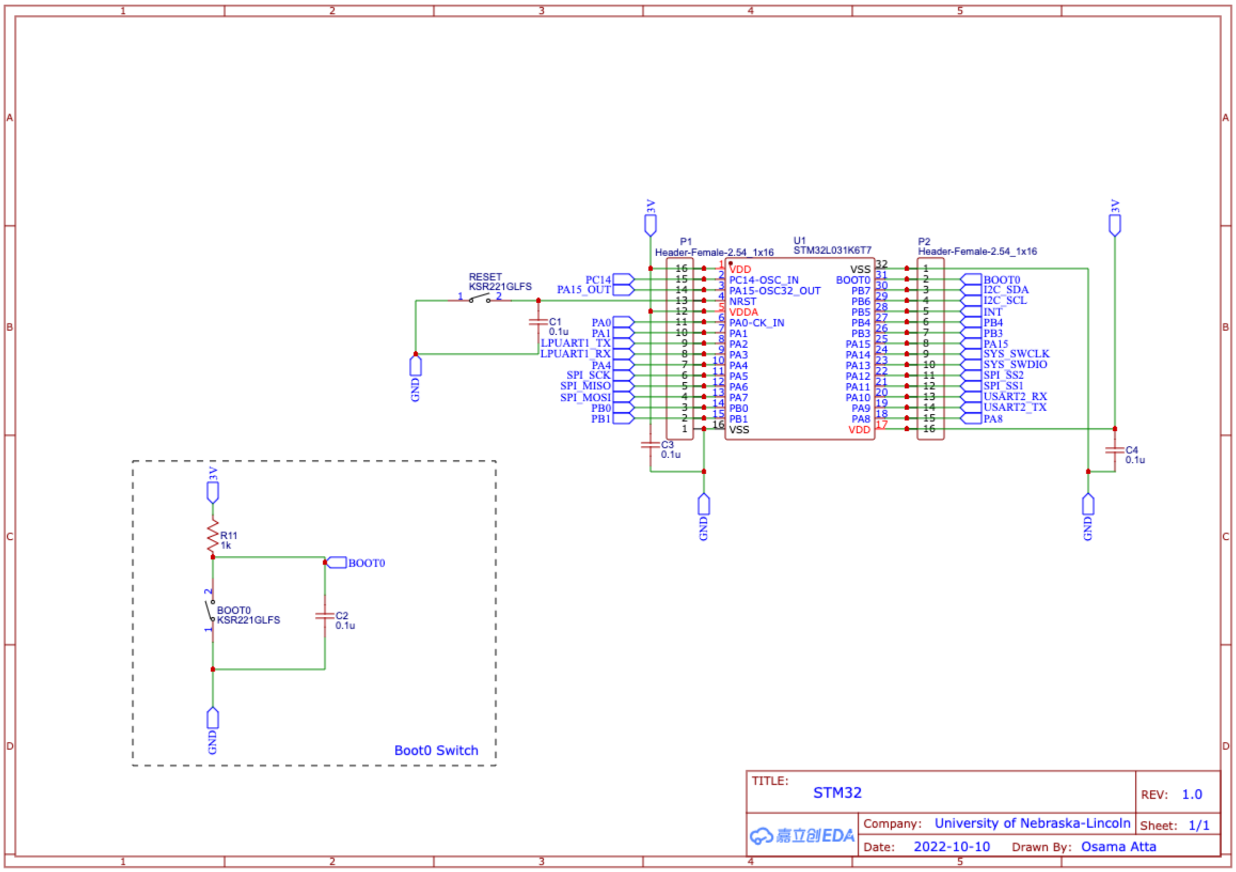

Figure 7: STM32 schematic along with the reset and BOOT0 circuitry

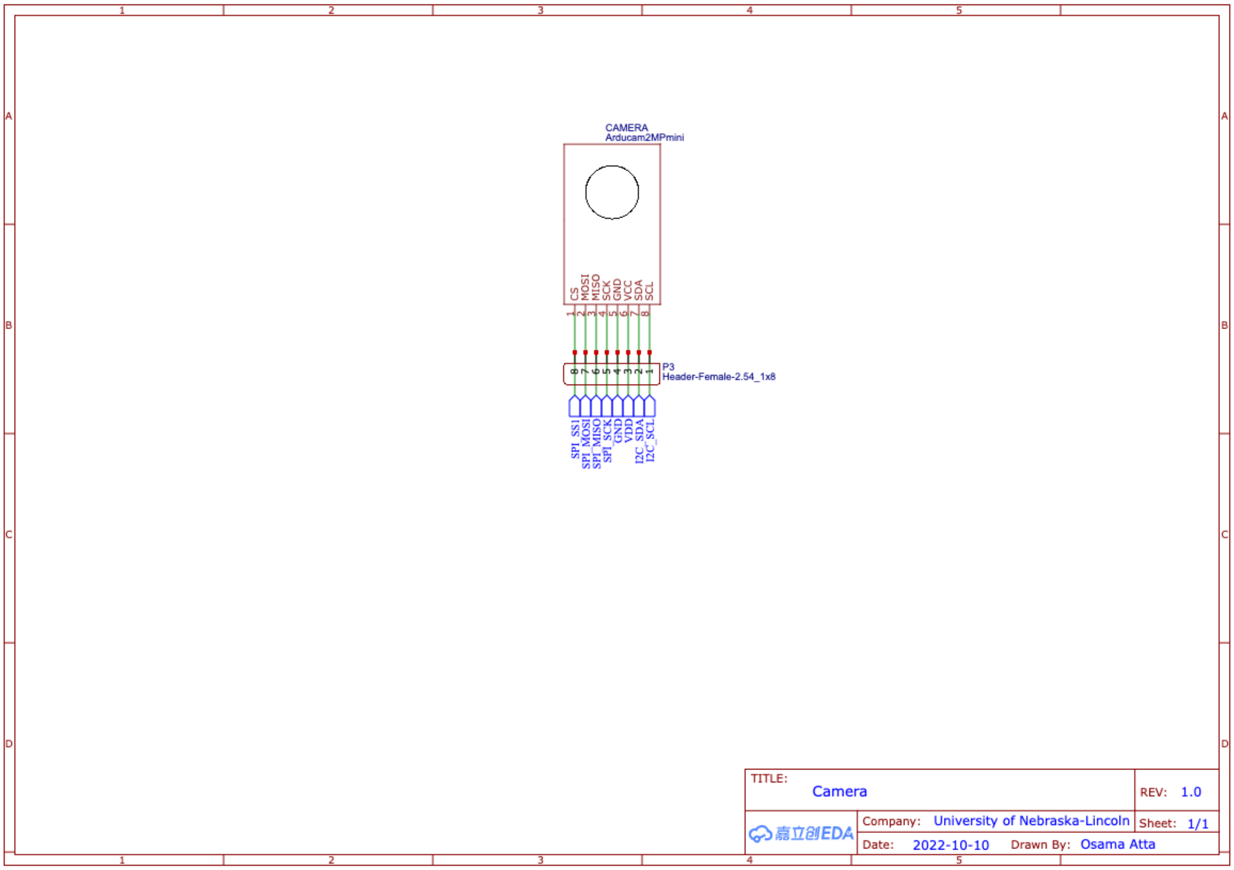

Figure 8: Camera schematic

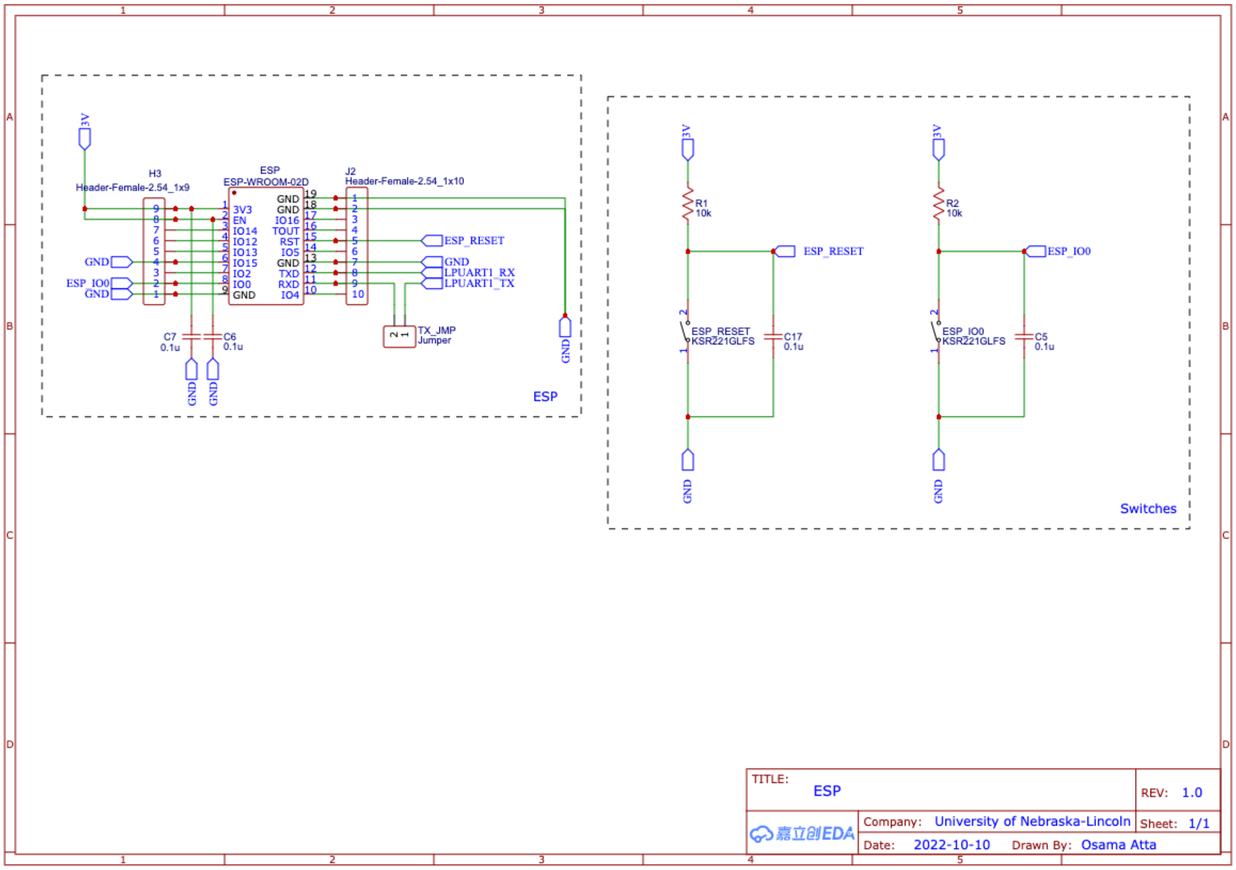

Figure 9: ESP schematic along with the reset and IO0 circuitry for the ESP

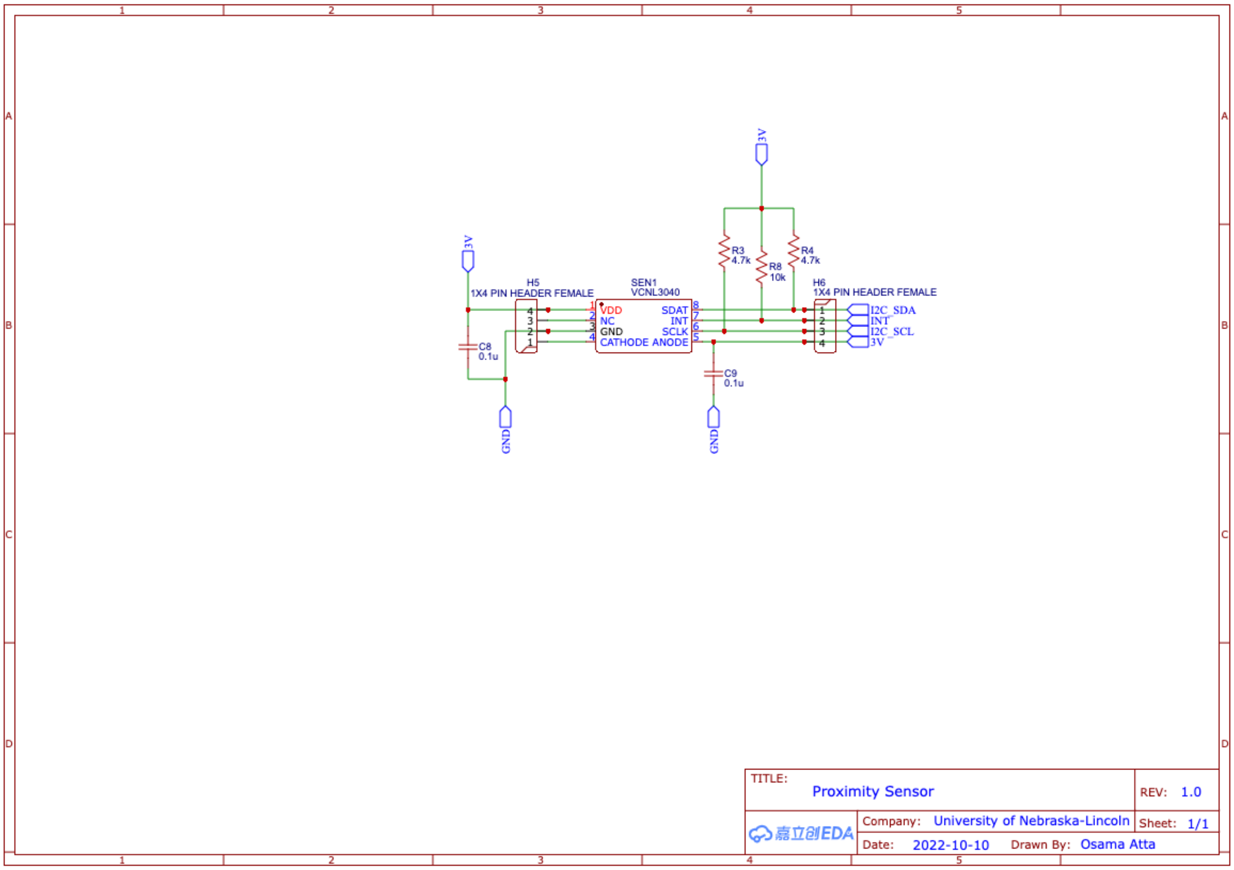

Figure 10: Proximity sensor schematic

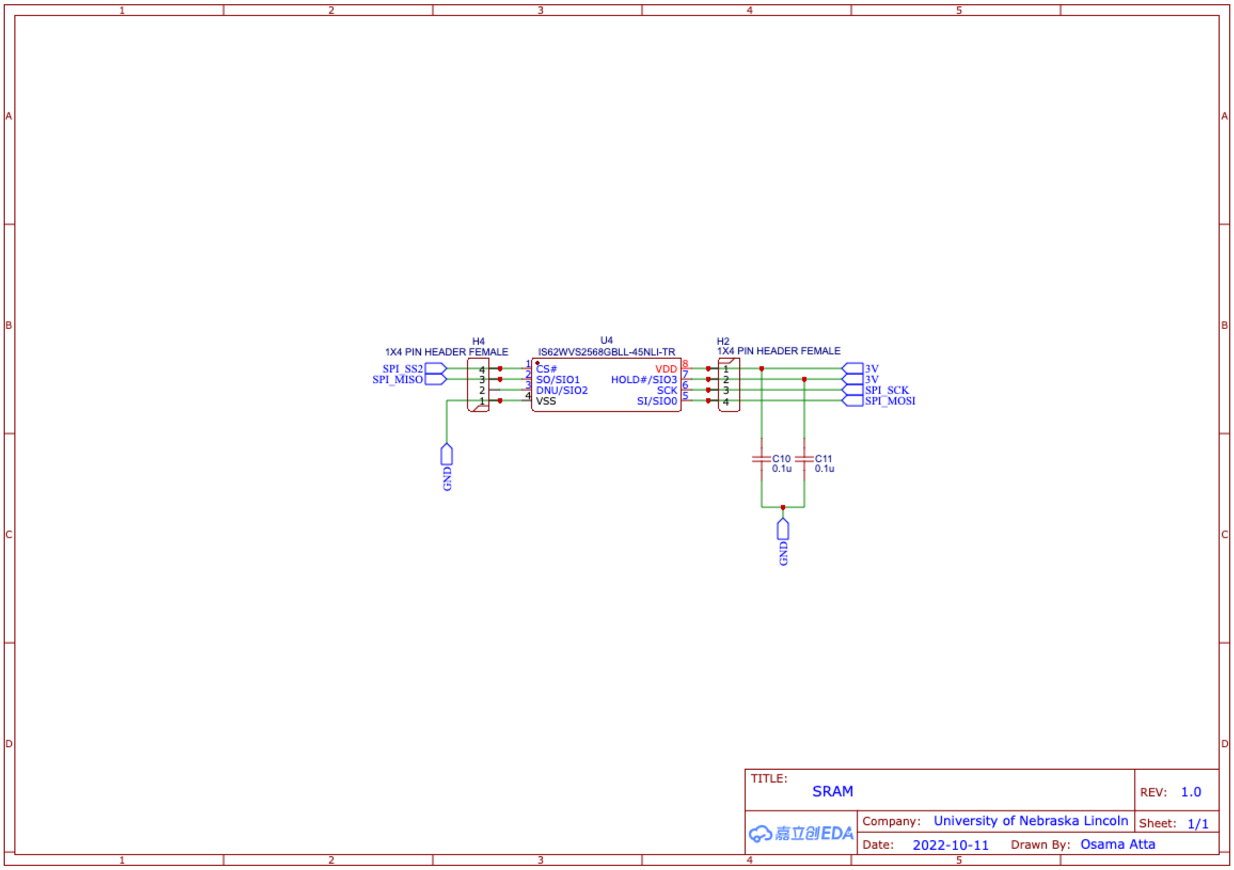

Figure 11: SRAM schematic

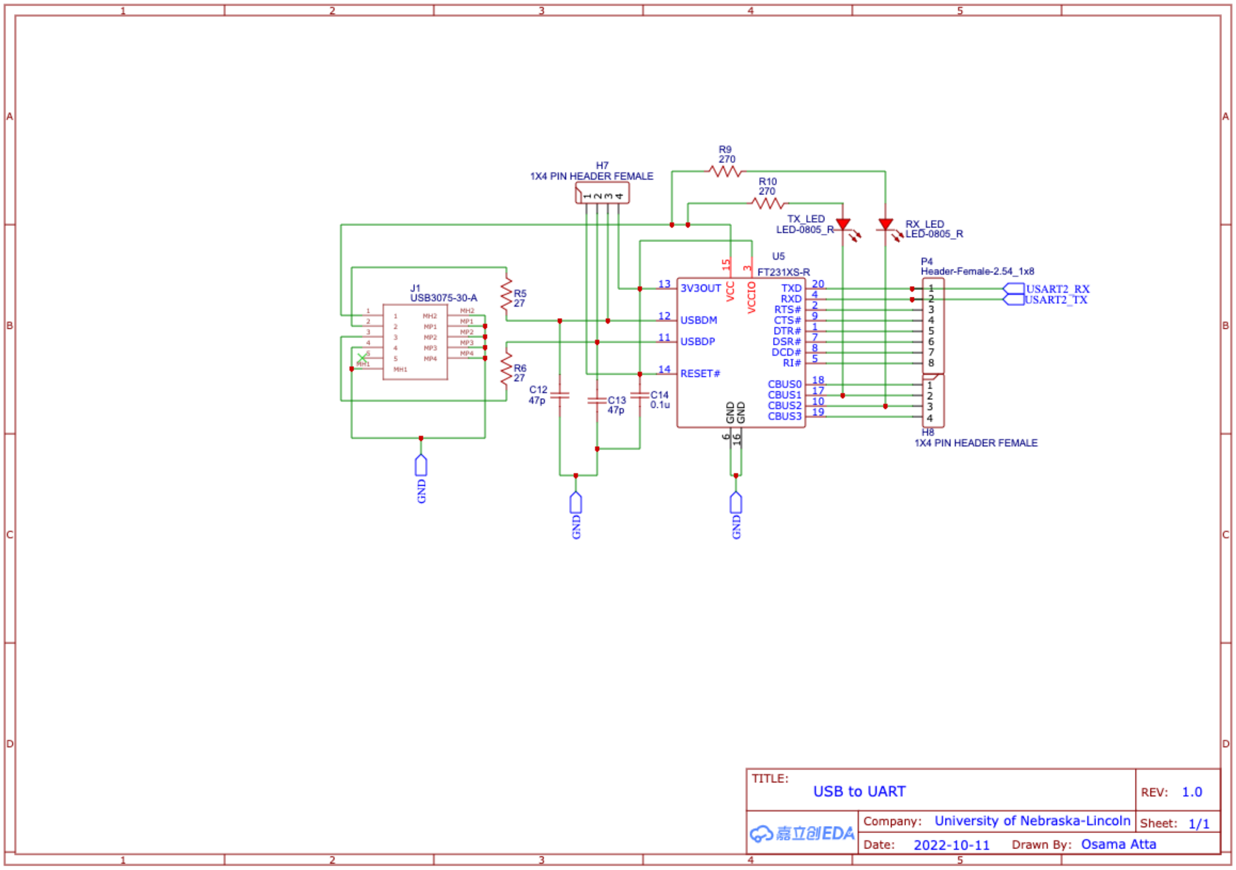

Figure 12: USB to UART schematic

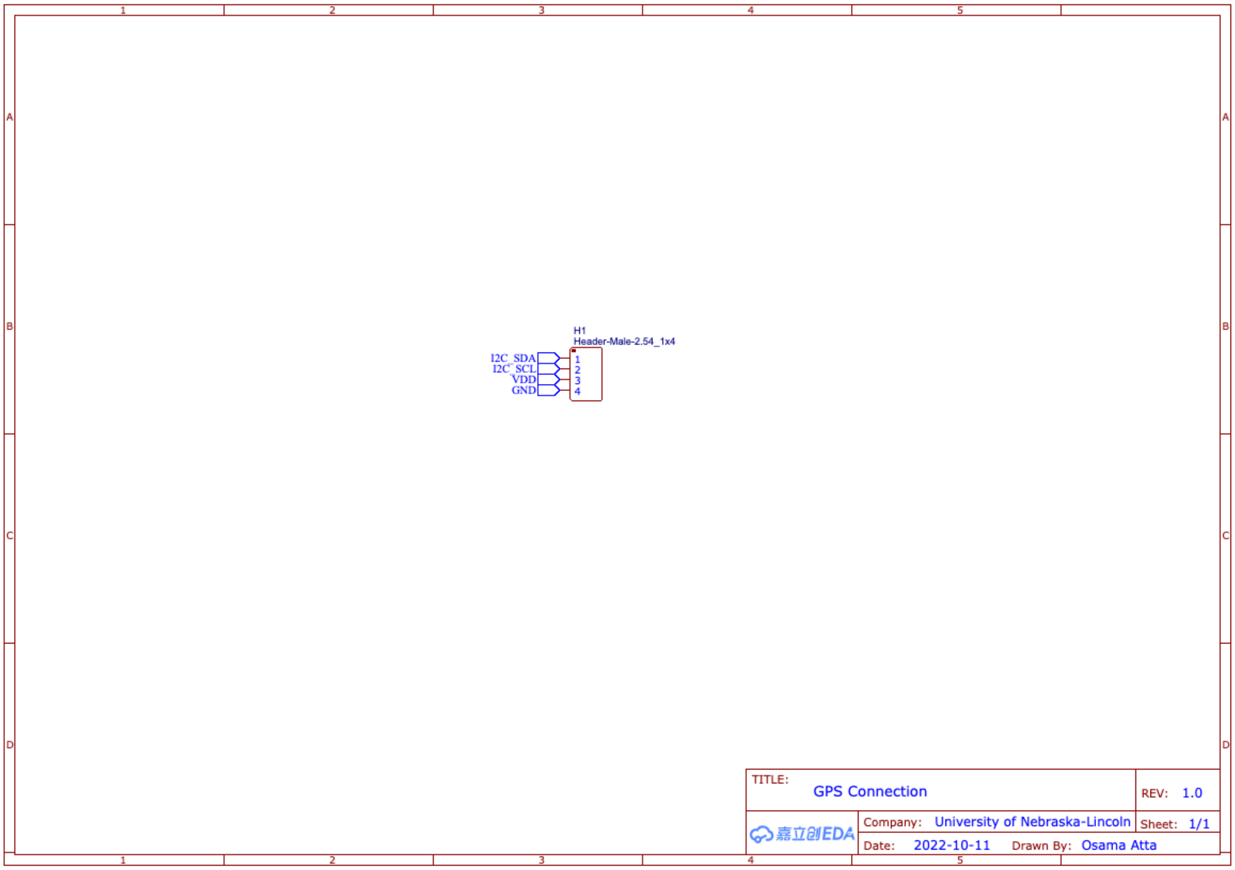

Figure 13: Schematic showing header used to connect to GPS module board

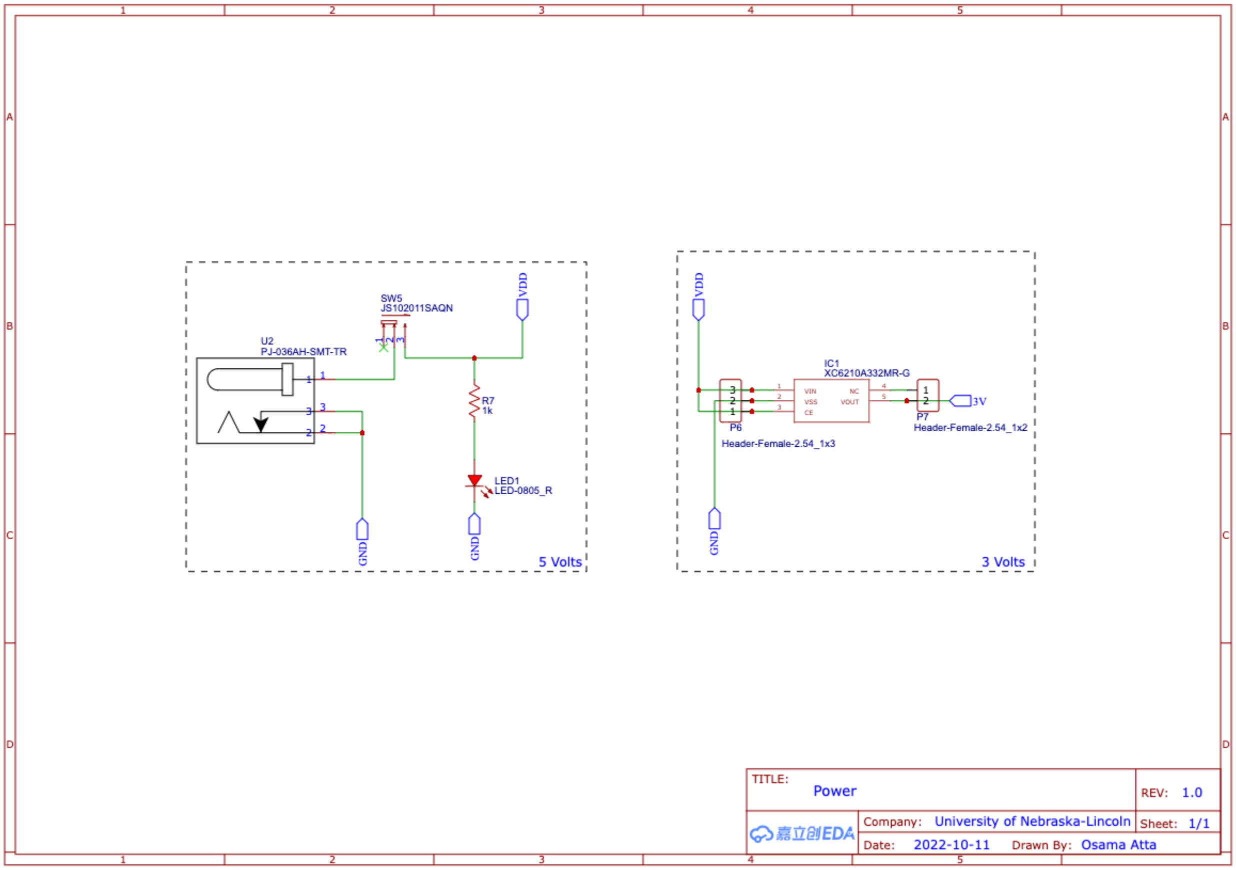

Figure 14: Schematic showing DC power jack and LDO voltage regulator

8.2 PCB Design

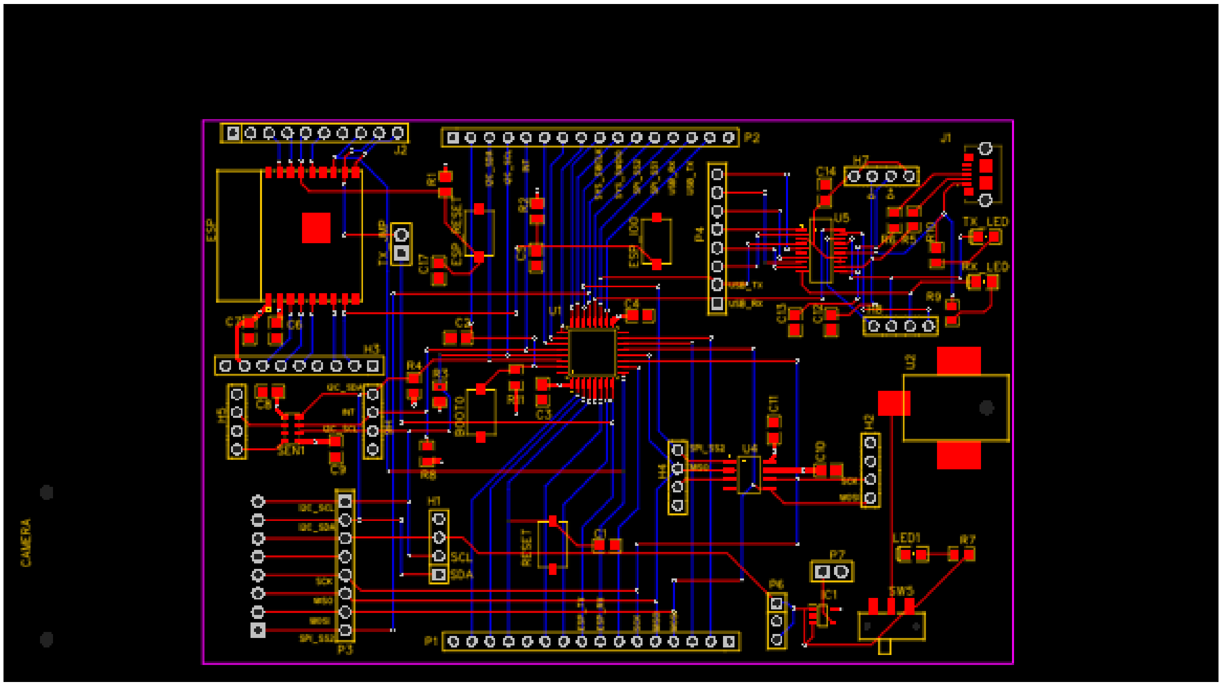

Figure 15: PCB connections without the GND and 3V planes to show the connections clearly

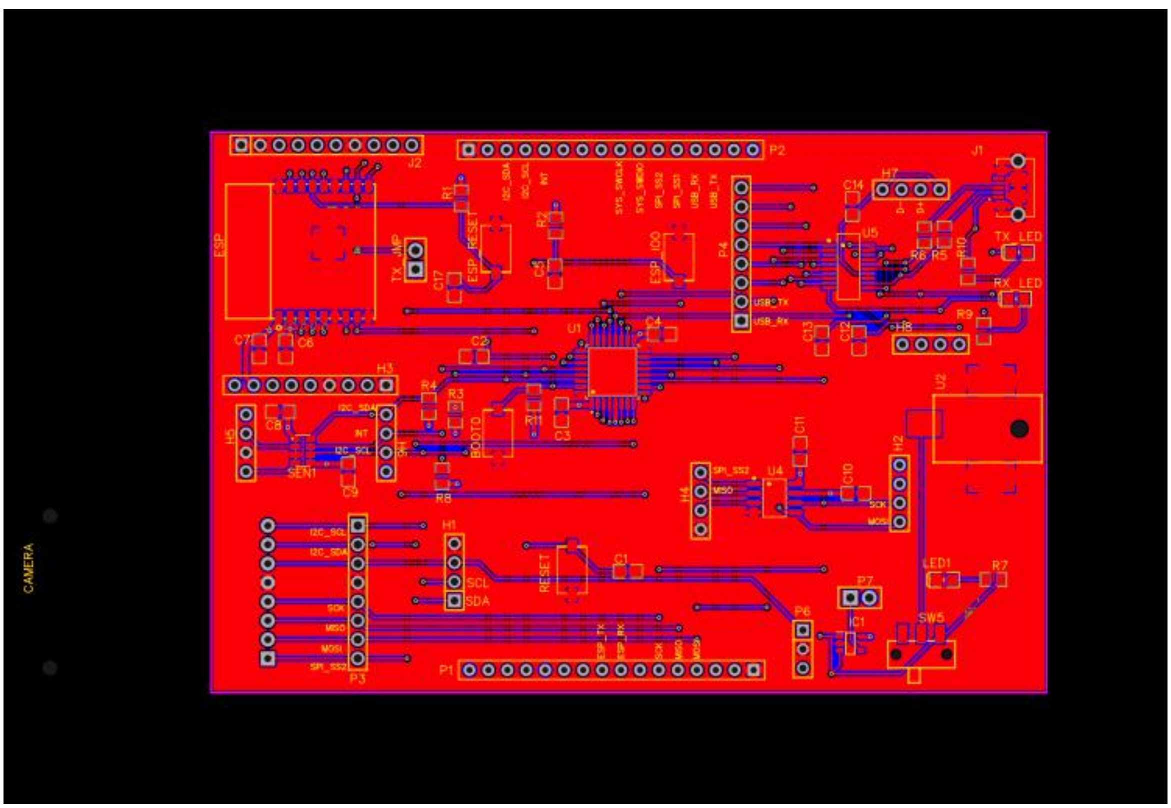

Figure 16: PCB with all layers

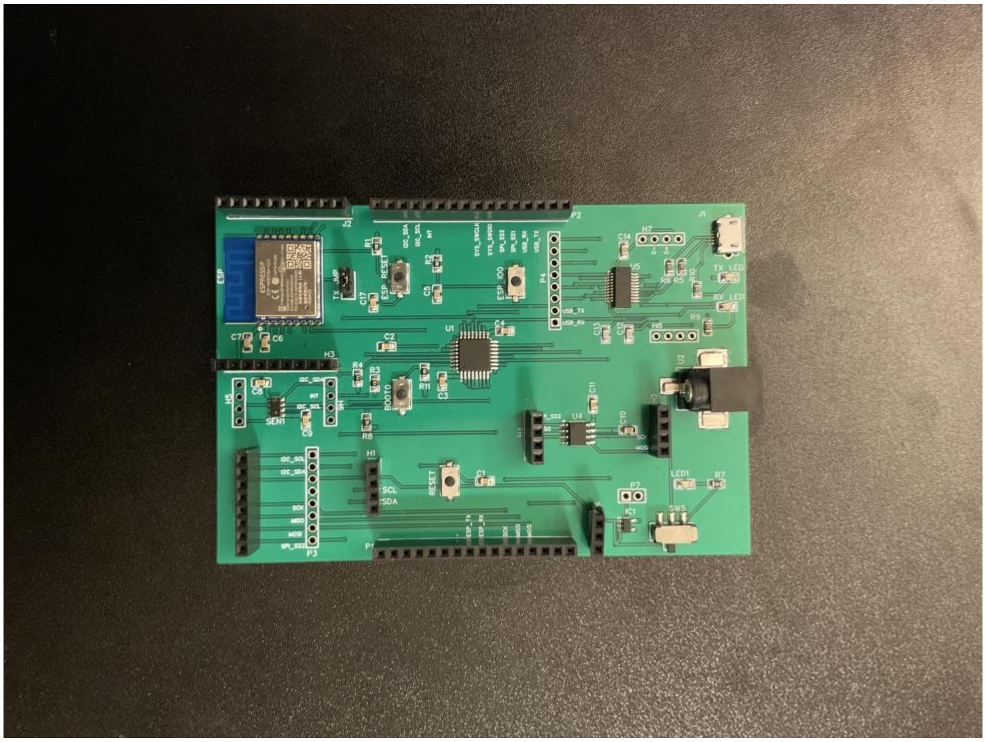

Figure 17: PCB will components soldered on

8.3 Bill of Materials

Part Name	Part Specification
USB3075-30-A	USB Connectors USB Connectors Micro B Skt, Bottom-SMT, R/A, 30u No Peg, W/shell stake, T&R
08055C104KAT4A	Multilayer Ceramic Capacitors MLCC - SMD/SMT Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 0.1uF X7R 0805 10%
XC6210A332MR-G	LDO Voltage Regulators LDO Voltage Regulators 700mA LDO
KSR221GLFS	Tactile Switches Tactile Switches Sub Min Tact SMT Switch
RC0805FR-071KL	Thick Film Resistors - SMD Thick Film Resistors - SMD 1 kOhms 62.5mW 0805 1%
ESP-WROOM-02D-N4	WiFi Modules - 802.11 WiFi Modules - 802.11 SMD Module, ESP8266EX, 32Mbits SPI flash, UART Mode, PCB antenna
CRCW08054K70JNEAC	Thick Film Resistors - SMD Thick Film Resistors - SMD 1/16watt 4.7Kohms 5%
RC0805FR-0710KL	Thick Film Resistors - SMD Thick Film Resistors - SMD 10 kOhms 62.5mW 0805 1%
SM0805UBWC	Standard LEDs - SMD Standard LEDs - SMD Ultra Blue 470 nm Water Clear
JS102011SAQN	Slide Switches Slide Switches 1PDT .3A Through Hole
PJ-036AH-SMT-TR	DC Power Connectors DC Power Connectors Power Jacks
VCNL3040	Proximity Sensors Proximity Sensors PROXIMITY SENSORS
FT231XS-R	USB Interface IC USB Interface IC USB to Full Serial UART IC SSOP-20
CL21B474KBFNNNF	Multilayer Ceramic Capacitors MLCC - SMD/SMT Multilayer Ceramic Capacitors MLCC - SMD/SMT 10V 0.47uF X5R 0805 10%
IS62WVS2568GBLL-45NLI-TR	SRAM SRAM 2Mb 256Kx8 45MHz 2.2-3.6V Serial SRAM
CC0805KRX5R6BB105	Multilayer Ceramic Capacitors MLCC - SMD/SMT Multilayer Ceramic Capacitors MLCC - SMD/SMT 10V 1uF X5R 0805 10%
OV2640 2 Mp module	Arducam Mini Module Camera Shield OV2640 2 Megapixels
CR0805-FX-27R0ELF	Thick Film Resistors - SMD 27ohm 1%
CR0805-FX-2700ELF	Thick Film Resistors - SMD 270ohm 1%
08055A470JAT2A	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 47pF C0G 0805 5%
STM32L031K6T7	Microcontroller

8.4 Demo Images

```
23:24:46.373 -> Hello World!
23:24:46.373 -> CAM for picture from camera
23:24:46.373 -> GPS for Coordinates
23:24:46.374 -> SEN for Proximity Sensor
23:24:46.374 -> RAM for a ram check
```

Figure 18: Initial shell menu showing user options

```
23:26:14.241 -> GPS
23:26:15.349 -> �,,,,0,00,99.99,,,,,,*48
23:26:15.349 -> �G�GSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30
23:26:15.349 -> �GPGSV,1,1,00*79
23:26:15.349 -> �GPGLL,,,,,,V,N*64
```

Figure 19: GPS Command output

```
23:27:01.326 -> Value from sensor: 0x39
23:27:01.437 -> Value from sensor: 0x39
23:27:01.518 -> Value from sensor: 0x38
23:27:01.623 -> Value from sensor: 0x37
23:27:01.756 -> Value from sensor: 0x35
23:27:01.849 -> Value from sensor: 0x33
23:27:01.981 -> Value from sensor: 0x31
23:27:02.070 -> Value from sensor: 0x32
23:27:02.180 -> Value from sensor: 0x31
23:27:02.277 -> Value from sensor: 0x31
23:27:02.376 -> Value from sensor: 0x31
23:27:02.474 -> Value from sensor: 0x31
23:27:02.598 -> Value from sensor: 0x31
23:27:02.689 -> Value from sensor: 0x31
23:27:02.782 -> Value from sensor: 0x39
```

Figure 20: Values being read from proximity sensor while object moves closer to the sensor

```
23:28:24.291 -> RAM
23:28:24.291 -> Input 2 hex value to check ram with:
23:28:26.800 -> Your input 0xff
23:28:26.800 -> Value in address 0xff
```

Figure 21: Successful RAM check operation with 0xFF being used as the test value

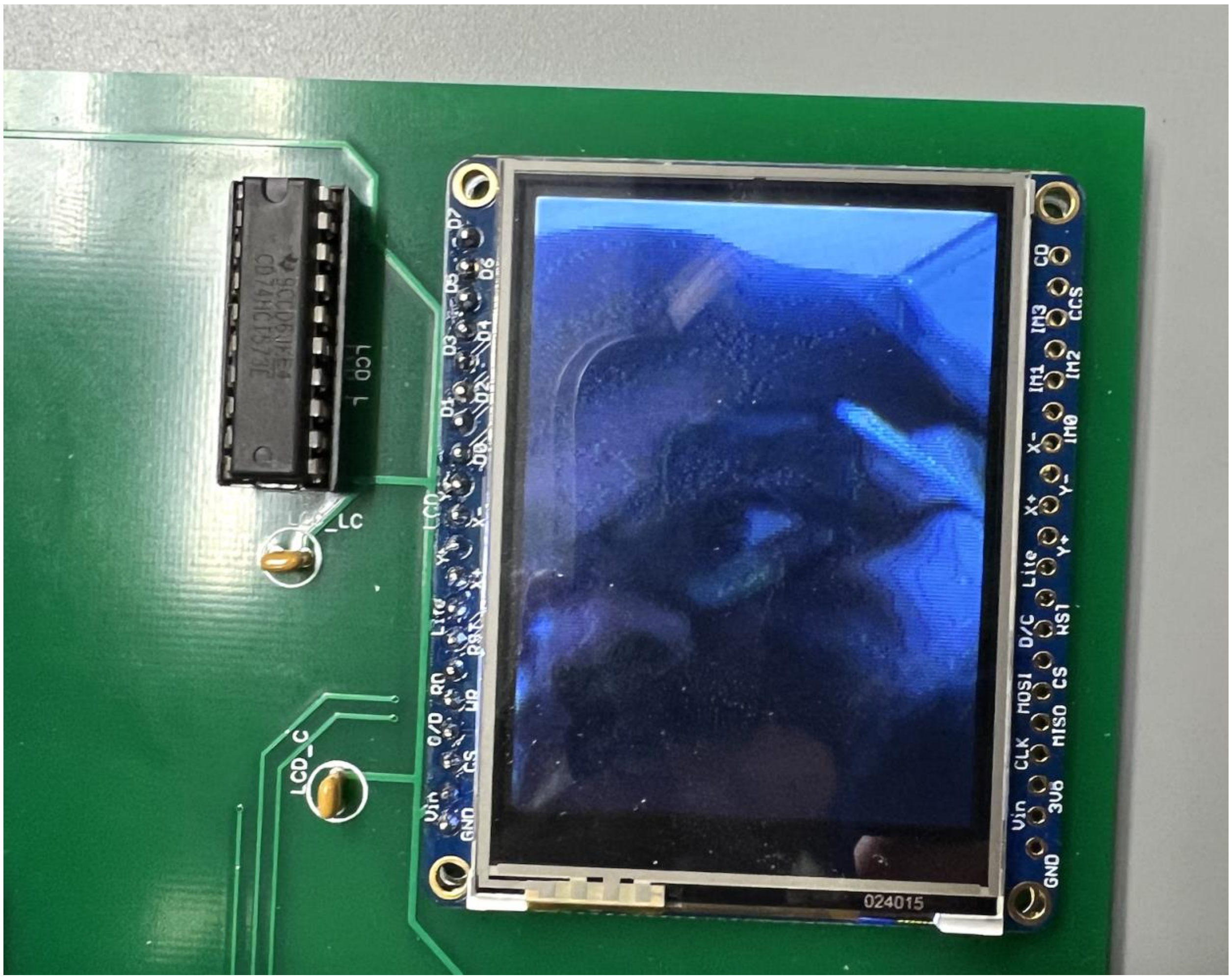

Figure 22: Successful image capture by the camera with image being displayed on the ECEN 433 board LCD

8.5 Implementation Code for Microcontroller

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */



/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

UART_HandleTypeDef hlpuart1;
UART_HandleTypeDef huart2;

SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */
//serial monitor printing related variables
const uint8_t configs[][2] = {
             {0xff, 0x0},
             {0x2c, 0xff},
             {0x2e, 0xdf},
             {0xff, 0x1},
             {0x3c, 0x32},
             {0x11, 0x0},
```

```
{0x09, 0x2},
{0x04, 0xa8},
{0x13, 0xe5},
{0x14, 0x48},
{0x2c, 0xc},
{0x33, 0x78},
{0x3a, 0x33},
{0x3b, 0xfb},
{0x3e, 0x0},
{0x43, 0x11},
{0x16, 0x10},
{0x39, 0x2},
{0x35, 0x88},
{0x22, 0xa},
{0x37, 0x40},
{0x23, 0x0},
{0x34, 0xa0},
{0x06, 0x2},
{0x06, 0x88},
{0x07, 0xc0},
{0x0d, 0xb7},
{0x0e, 0x1},
{0x4c, 0x0},
{0x4a, 0x81},
{0x21, 0x99},
{0x24, 0x40},
{0x25, 0x38},
{0x26, 0x82},
{0x5c, 0x0},
{0x63, 0x0},
{0x46, 0x22},
{0x0c, 0x3a},
{0x5d, 0x55},
{0x5e, 0x7d},
{0x5f, 0x7d},
{0x60, 0x55},
{0x61, 0x70},
{0x62, 0x80},
{0x7c, 0x5},
{0x20, 0x80},
{0x28, 0x30},
{0x6c, 0x0},
{0x6d, 0x80},
{0x6e, 0x0},
{0x70, 0x2},
{0x71, 0x94},
{0x73, 0xc1},
{0x3d, 0x34},
{0x12, 0x4},
{0x5a, 0x57},
{0x4f, 0xbb},
{0x50, 0x9c},
{0xff, 0x0},
{0xe5, 0x7f},
{0xf9, 0xc0},
{0x41, 0x24},
{0xe0, 0x14},
{0x76, 0xff},
{0x33, 0xa0},
{0x42, 0x20},
{0x43, 0x18},
```

```
{0x4c, 0x0},
{0x87, 0xd0},
{0x88, 0x3f},
{0xd7, 0x3},
{0xd9, 0x10},
{0xd3, 0x82},
{0xc8, 0x8},
{0xc9, 0x80},
{0x7c, 0x0},
{0x7d, 0x0},
{0x7c, 0x3},
{0x7d, 0x48},
{0x7d, 0x48},
{0x7c, 0x8},
{0x7d, 0x20},
{0x7d, 0x10},
{0x7d, 0xe},
{0x90, 0x0},
{0x91, 0xe},
{0x91, 0x1a},
{0x91, 0x31},
{0x91, 0x5a},
{0x91, 0x69},
{0x91, 0x75},
{0x91, 0x7e},
{0x91, 0x88},
{0x91, 0x8f},
{0x91, 0x96},
{0x91, 0xa3},
{0x91, 0xaf},
{0x91, 0xc4},
{0x91, 0xd7},
{0x91, 0xe8},
{0x91, 0x20},
{0x92, 0x0},
{0x93, 0x6},
{0x93, 0xe3},
{0x93, 0x3},
{0x93, 0x3},
{0x93, 0x0},
{0x93, 0x2},
{0x93, 0x0},
{0x93, 0x0},
{0x93, 0x0},
{0x93, 0x0},
{0x93, 0x0},
{0x93, 0x0},
{0x93, 0x0},
{0x96, 0x0},
{0x97, 0x8},
{0x97, 0x19},
{0x97, 0x2},
{0x97, 0xc},
{0x97, 0x24},
{0x97, 0x30},
{0x97, 0x28},
{0x97, 0x26},
{0x97, 0x2},
{0x97, 0x98},
{0x97, 0x80},
{0x97, 0x0},
```

```
{0x97, 0x0},
{0xa4, 0x0},
{0xa8, 0x0},
{0xc5, 0x11},
{0xc6, 0x51},
{0xbf, 0x80},
{0xc7, 0x10},
{0xb6, 0x66},
{0xb8, 0xa5},
{0xb7, 0x64},
{0xb9, 0x7c},
{0xb3, 0xaf},
{0xb4, 0x97},
{0xb5, 0xff},
{0xb0, 0xc5},
{0xb1, 0x94},
{0xb2, 0xf},
{0xc4, 0x5c},
{0xa6, 0x0},
{0xa7, 0x20},
{0xa7, 0xd8},
{0xa7, 0x1b},
{0xa7, 0x31},
{0xa7, 0x0},
{0xa7, 0x18},
{0xa7, 0x20},
{0xa7, 0xd8},
{0xa7, 0x19},
{0xa7, 0x31},
{0xa7, 0x0},
{0xa7, 0x18},
{0xa7, 0x20},
{0xa7, 0xd8},
{0xa7, 0x19},
{0xa7, 0x31},
{0xa7, 0x0},
{0xa7, 0x18},
{0x7f, 0x0},
{0xe5, 0x1f},
{0xe1, 0x77},
{0xdd, 0x7f},
{0xc2, 0xe},
{0xff, 0x0},
{0xe0, 0x4},
{0xc0, 0xc8},
{0xc1, 0x96},
{0x86, 0x3d},
{0x51, 0x90},
{0x52, 0x2c},
{0x53, 0x0},
{0x54, 0x0},
{0x55, 0x88},
{0x57, 0x0},
{0x50, 0x92},
{0x5a, 0x50},
{0x5b, 0x3c},
{0x5c, 0x0},
{0xd3, 0x4},
{0xe0, 0x0},
{0xff, 0x0},
{0x05, 0x0},
```

```
              {0xda, 0x8},
              {0xd7, 0x3},
              {0xe0, 0x0},
              {0x05, 0x0}         };


//SRAM related variabl
const uint8_t SRAM_WRMR = 0x01;
const uint8_t SRAM_SEQ = 0x00;



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_LPUART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_SPI1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

unsigned int charToHex(char val){
       if (val == '0'){
              return 0x0;
       }
       else if (val == '1'){
              return 0x1;
       }
       else if (val == '2'){
              return 0x2;
       }
       else if (val == '3'){
              return 0x3;
       }
       else if (val == '4'){
              return 0x4;
       }
       else if (val == '5'){
              return 0x5;
       }
       else if (val == '6'){
              return 0x6;
       }
       else if (val == '7'){
              return 0x7;
       }
       else if (val == '8'){
              return 0x8;
       }
       else if (val == '9'){
              return 0x9;
       }
       else if (val == 'A'){
              return 0xA;
```

```
	}
	else if (val == 'B'){
		return 0xB;
	}
	else if (val == 'C'){
		return 0xC;
	}
	else if (val == 'D'){
		return 0xD;
	}
	else if (val == 'E'){
		return 0xE;
	}
	else{
		return 0xF;
	}
}
/*=====================================================================================
=============*/
HAL_StatusTypeDef printToMonitor(uint8_t *data, uint8_t size){
	HAL_StatusTypeDef ret;
	uint8_t printError[16] = "Error Printing";

	ret = HAL_UART_Transmit(&huart2, data, size, HAL_MAX_DELAY);
	if (ret != HAL_OK){
		ret = HAL_UART_Transmit(&huart2, printError, sizeof(printError)-1,
HAL_MAX_DELAY);
		return ret;
	}

	return ret;
}
/*=====================================================================================
=============*/
HAL_StatusTypeDef sensor_init(){

	HAL_StatusTypeDef ret;

	int8_t configData[3];

	configData[0]=0x04; //command code
	configData[1]=0x00;
	configData[2]=0x03;

	ret = HAL_I2C_Master_Transmit(&hi2c1, SENSORADDRESS<<1, (uint8_t *)&configData,
3, 2000);
	if (ret != HAL_OK)
	{
		return ret;
	}

	configData[0] = 0x03; //command code
	configData[1] = 0x40;
	configData[2] = 0x08;

	ret = HAL_I2C_Master_Transmit(&hi2c1, SENSORADDRESS<<1, (uint8_t *)&configData,
3, 2000);
	if (ret != HAL_OK)
	{
		return ret;
	}
```

```
	return ret;
}
/*=====================================================================================
=============*/
HAL_StatusTypeDef getSensorValue(uint8_t *data){

	HAL_StatusTypeDef ret;
	uint8_t sensorDataBuffer[2];

	ret = HAL_I2C_Mem_Read(&hi2c1,SENSORADDRESS<<1,0x08,1,(uint8_t
*)&sensorDataBuffer,2,HAL_MAX_DELAY);
	if (ret != HAL_OK)
		{
			return ret;
		}

	if (sensorDataBuffer[0]<6){
		*data=0x39;
	}else if(sensorDataBuffer[0]<9 && sensorDataBuffer[1] == 0){
		*data=0x38;
	}else if(sensorDataBuffer[0]<35 && sensorDataBuffer[1] == 0){
		*data=0x37;
	}else if(sensorDataBuffer[0]<65 && sensorDataBuffer[1] == 0){
		*data = 0x36;
	}else if(sensorDataBuffer[0]<90 && sensorDataBuffer[1] == 0){
		*data = 0x35;
	}else if(sensorDataBuffer[0]<125 && sensorDataBuffer[1] == 0){
		*data = 0x34;
	}else if(sensorDataBuffer[0]<175 && sensorDataBuffer[1] == 0){
		*data = 0x33;
	}else if(sensorDataBuffer[0] < 225 && sensorDataBuffer[1] == 0){
		*data = 0x32;
	}else{
		*data = 0x31;
	}

	return ret;
}
/*=====================================================================================
=============*/
HAL_StatusTypeDef getGpsData(){
	HAL_StatusTypeDef ret;
	int8_t gpsDataBuffer[256];
	int8_t init = 0xFF;

	ret = HAL_I2C_Master_Transmit(&hi2c1, GPSADDRESS<<1, (uint8_t *)&init, 1,
HAL_MAX_DELAY);
		if (ret != HAL_OK)
		{
			return ret;
		}
	HAL_Delay(40);

	do{
		ret = HAL_I2C_Master_Receive(&hi2c1, GPSADDRESS<<1, (uint8_t
*)&gpsDataBuffer, 256, HAL_MAX_DELAY);
			if (ret != HAL_OK){
				return ret;
			}
		//HAL_Delay(40);
```

```
	}while(gpsDataBuffer[0] == -1);

	HAL_UART_Transmit(&huart2,(uint8_t *)&gpsDataBuffer,162,HAL_MAX_DELAY);

	return ret;
}
/*=====================================================================================
=============*/
void ram_init(){

	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET);
	HAL_SPI_Transmit(&hspi1, (uint8_t *)&SRAM_WRMR, 1, 100);
	HAL_SPI_Transmit(&hspi1, (uint8_t *)&SRAM_SEQ, 1, 100);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET);
}
/*=====================================================================================
=============*/
void ram_write_split(uint8_t high, uint8_t med, uint8_t low, uint8_t value){

	static uint8_t srcBuff[5];

	srcBuff[0] = 0x02;			// write command code
	srcBuff[1] = high; //high address
	srcBuff[2] = med; //middle address
	srcBuff[3] = low; //low address
	srcBuff[4] = value; //data

	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET); //slave select low
    HAL_SPI_Transmit(&hspi1, (uint8_t *)&srcBuff, 5, 100); //command -> addr -> data
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET); //slave select high

}
/*=====================================================================================
=============*/
uint8_t ram_read_split(uint8_t high, uint8_t med, uint8_t low, uint8_t *data){

	static uint8_t srcReadBuff[4];

	srcReadBuff[0] = 0x03;		//Read Command Code
	srcReadBuff[1] = high;	//high address
	srcReadBuff[2] = med;	//middle address
	srcReadBuff[3] = low;	//low address

	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET);//slave select low

	HAL_SPI_Transmit(&hspi1, (uint8_t *)&srcReadBuff, 4, 100); //command -> addr
	HAL_SPI_Receive(&hspi1, data, 1, 100); //they are definitely trolling with this


	HAL_SPI_Receive(&hspi1, data, 1, 100); //how should we retun the data? -> one
by one
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET); //slave select high

	return *data;

}
/*=====================================================================================
=============*/
HAL_StatusTypeDef cam_config(){
	uint8_t i = 0;
```

```
	HAL_StatusTypeDef ret;
	uint8_t printError[16];
	int8_t configData[2];


	configData[0] = 0xff;
	configData[1] = 0x01;
	ret = HAL_I2C_Master_Transmit(&hi2c1, CAMADDRESSWRITE, (uint8_t *)&configData, 
2, 100);
	if (ret != HAL_OK){
		return ret;
	}


	configData[0] = 0x12;
	configData[1] = 0x80;
	ret = HAL_I2C_Master_Transmit(&hi2c1, 0x60, (uint8_t *)&configData, 2, 100);
	if (ret != HAL_OK){
		return ret;
	}


	HAL_Delay(200);

	for(i = 0; i<193; i++){
		ret = HAL_I2C_Master_Transmit(&hi2c1, 0x60, (uint8_t *)&configs[i], 2, 
100);
		if (ret != HAL_OK){
			return ret;
		}
	}

	return ret;
}
/*=====================================================================================
=============*/
void esp_tx(uint8_t data){
	uint8_t txbuff[1];
	txbuff[0] = data;
	HAL_UART_Transmit(&hlpuart1,(uint8_t *) &txbuff, 1,HAL_MAX_DELAY);
}
/*=====================================================================================
=============*/
uint8_t esp_rx(){
	uint8_t rxbuff[1];
	HAL_UART_Receive(&hlpuart1,(uint8_t *)&rxbuff, sizeof(rxbuff),HAL_MAX_DELAY);
	return rxbuff[0];
}
/*=====================================================================================
=============*/
void cam_pic(void){

	uint8_t srcBuff[2];
	char uart_buf_cam[50];
	int uart_buf_len_cam;
	uint8_t readysend;
	uint8_t readyrec;
	uint8_t srcTestBuff = 0x3C;
	uint8_t high = 0x00;
	uint8_t low = 0x00;
	uint8_t cover = 0x00;
```

```
	srcBuff[0] = 0x84; //msb is 1 for write
	srcBuff[1]=  0x20; // try 0x33


	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET); //slave select low
	HAL_SPI_Transmit(&hspi1, (uint8_t *)&srcBuff, 2, HAL_MAX_DELAY); //setup
register
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high


	srcBuff[1]=  0x10;

	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET); //slave select low
	HAL_SPI_Transmit(&hspi1, (uint8_t *)&srcBuff, 2, HAL_MAX_DELAY); //setup
register
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high



	srcBuff[1] = 0x02;

	//first capture
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET); //slave select low
	HAL_SPI_Transmit(&hspi1, (uint8_t *)&srcBuff, 2, HAL_MAX_DELAY); //start the
camera capture
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high


	readysend=0x41;

	do{
		HAL_Delay(10);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET); //slave select low
		HAL_SPI_Transmit(&hspi1,&readysend, 1, HAL_MAX_DELAY);
		HAL_SPI_Receive(&hspi1,&readyrec, 1, HAL_MAX_DELAY);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high
	}while((readyrec & 0x08) == 0);


	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high

	uart_buf_len_cam = sprintf(uart_buf_cam, "Picture taken!\n");
	printToMonitor((uint8_t *)&uart_buf_cam, uart_buf_len_cam);

	srcTestBuff = 0x3C;


	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_RESET); //slave select low


	HAL_SPI_Transmit(&hspi1,&srcTestBuff, 1, HAL_MAX_DELAY);
	HAL_SPI_Receive(&hspi1, &high, 1, HAL_MAX_DELAY); //dummy bit

	HAL_UART_Transmit(&hlpuart1,(uint8_t *) "IMG", 3,HAL_MAX_DELAY);
	HAL_UART_Transmit(&hlpuart1,(uint8_t *) &cover, 1,HAL_MAX_DELAY);
	HAL_UART_Transmit(&hlpuart1,(uint8_t *) &cover, 1,HAL_MAX_DELAY);

	for(uint8_t t = 0; t < 240; t++){
			for(uint16_t v = 0; v < 320; v++){
```

```
                                HAL_SPI_Receive(&hspi1, &high, 1, HAL_MAX_DELAY);
                                HAL_SPI_Receive(&hspi1, &low, 1, HAL_MAX_DELAY);

                                if((t >= 40 && t < 200) && (v >= 100 && v < 220)){

                                        HAL_UART_Transmit(&hlpuart1,&low, 1,HAL_MAX_DELAY);
                                        HAL_UART_Transmit(&hlpuart1,&high, 1,HAL_MAX_DELAY);

                                }
                        }
                }

        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET); //slave select high
}
/*=====================================================================================
=============*/

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  char uart_buf[150];
  int uart_buf_len;


  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_LPUART1_UART_Init();
  MX_USART2_UART_Init();
  MX_SPI1_Init();
  /* USER CODE BEGIN 2 */
  uint8_t didntwork[4] = "FAIL\n";
  uint8_t invalidInput[13] = "Invalid input";
  HAL_StatusTypeDef checker;
```

```
  uint8_t ramBuff = 0;
  uint8_t espbuff;
  uint8_t sensorValue;


  uint8_t high;
  uint8_t low;

  uint8_t ramCheck;



  //we need to initalize the slave select pins here
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_12, GPIO_PIN_SET);//slave select 2
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, GPIO_PIN_SET);//slave select 1


  checker = sensor_init();
  if(checker != HAL_OK){
         uart_buf_len = sprintf(uart_buf, "sensor init didnt work\n");
         printToMonitor((uint8_t *)&uart_buf, uart_buf_len);
   }

  ram_init();

  checker = cam_config();
  if(checker != HAL_OK){
         uart_buf_len = sprintf(uart_buf, "cam config didnt work\n");
         printToMonitor((uint8_t *)&uart_buf, uart_buf_len);
  }

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

//*************************************************************************************
*****
         uart_buf_len = sprintf(uart_buf,
                        "Hello World!\nCAM for picture from camera\nGPS for
Coordinates\nSEN for Proximity Sensor\nRAM for a ram check\n");

         printToMonitor((uint8_t *)&uart_buf, uart_buf_len);

         uint8_t response[4] = {0,0,0,0}; //check with board

         HAL_UART_Receive(&huart2, (uint8_t *)&response, sizeof(response),
HAL_MAX_DELAY);

         uart_buf_len = sprintf(uart_buf,
                               "%c%c%c\r\n",response[0],response[1],response[2]);

         printToMonitor((uint8_t *)&uart_buf, uart_buf_len);
```

```
        if(response[0] == 'G' && response[1] == 'P' && response[2] == 'S'){
              checker = getGpsData();
              if (checker != HAL_OK){

HAL_UART_Transmit(&huart2,didntwork,sizeof(didntwork),HAL_MAX_DELAY);
              }
        }
        else if(response[0] == 'C' && response[1] == 'A' && response[2] == 'M'){


              uart_buf_len = sprintf(uart_buf, "Say Cheese!\n");
              printToMonitor((uint8_t *)&uart_buf, uart_buf_len);


              do{
                    espbuff = esp_rx();

                    uart_buf_len = sprintf(uart_buf,
                                                   "value we are getting from esp:
%c\n", espbuff);
                                printToMonitor((uint8_t *)&uart_buf, uart_buf_len);


              }while(espbuff != 'R');


              cam_pic();


              uart_buf_len = sprintf(uart_buf,
                                          "transmitted!\n");
                    printToMonitor((uint8_t *)&uart_buf, uart_buf_len);

             }
        else if(response[0] == 'S' && response[1] == 'E' && response[2] == 'N'){


              while(1){

                    getSensorValue(&sensorValue);

                    uart_buf_len = sprintf(uart_buf,
                                                         "Value from sensor:
0x%x\n", sensorValue);
                                      printToMonitor((uint8_t *)&uart_buf,
uart_buf_len);

                    HAL_Delay(100);

              }



        }

        else if(response[0] == 'R' && response[1] == 'A' && response[2] == 'M'){

              response[4] = 0;

              uart_buf_len = sprintf(uart_buf,"Input 2 hex value to check ram
with:\n");
```

```
                                                    printToMonitor((uint8_t
*)&uart_buf, uart_buf_len);

              HAL_UART_Receive(&huart2, (uint8_t *)&response, 3, HAL_MAX_DELAY);

              high = charToHex(response[0]);
              low = charToHex(response[1]);

              ramCheck = (high << 4) | (low) ;

              uart_buf_len = sprintf(uart_buf,"Your input 0x%x\n", ramCheck);
             printToMonitor((uint8_t *)&uart_buf, uart_buf_len);


             ram_write_split(0x10,0xFF,0xFF, ramCheck);

             ramBuff = 0;

             ram_read_split(0x10,0xFF,0xFF,&ramBuff);

             if(ramBuff != ramCheck){
                   uart_buf_len = sprintf(uart_buf,"Ram Check Failed\n");
                                printToMonitor((uint8_t *)&uart_buf, uart_buf_len);

             }

             uart_buf_len = sprintf(uart_buf,"Value in address 0x%x\n", ramBuff);
             printToMonitor((uint8_t *)&uart_buf, uart_buf_len);

        }


        else{

HAL_UART_Transmit(&huart2,invalidInput,sizeof(invalidInput),HAL_MAX_DELAY);
        }



//*************************************************************************************
*****

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_LPUART1
                              |RCC_PERIPHCLK_I2C1;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_PCLK1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00000708;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
```

```
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief LPUART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_LPUART1_UART_Init(void)
{

  /* USER CODE BEGIN LPUART1_Init 0 */

  /* USER CODE END LPUART1_Init 0 */

  /* USER CODE BEGIN LPUART1_Init 1 */

  /* USER CODE END LPUART1_Init 1 */
  hlpuart1.Instance = LPUART1;
  hlpuart1.Init.BaudRate = 115200;
  hlpuart1.Init.WordLength = UART_WORDLENGTH_8B;
  hlpuart1.Init.StopBits = UART_STOPBITS_1;
  hlpuart1.Init.Parity = UART_PARITY_NONE;
  hlpuart1.Init.Mode = UART_MODE_TX_RX;
  hlpuart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  hlpuart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  hlpuart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&hlpuart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN LPUART1_Init 2 */

  /* USER CODE END LPUART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, SPI1_SS1_Pin|SPI1_SS2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : SPI1_SS1_Pin SPI1_SS2_Pin */
  GPIO_InitStruct.Pin = SPI1_SS1_Pin|SPI1_SS2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : INT_Pin */
  GPIO_InitStruct.Pin = INT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(INT_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
```

```
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```