# a07g-exploring-the-CLI

- Team Number: 19
- Team Name: AC-DC
- Team Members: Madison Hughes and Aditya Rangamani
- GitHub Repository URL: https://github.com/ese5160/final-project-a07g-a14g-t19-ac-dc.git
- Description of test hardware: (development boards, sensors, actuators, laptop + OS, etc) SAMW25, Saelae Logic 8, Laptop+Windows, Microchip Studio, PuTTY

# Software Architecture

# Hardware Requirements Specification (HRS)

HRS 01 - SAMW25 shall be our main microcontroller for all processing. The SAMD21 will be our microcontroller and the WINC1500 WiFi chip will facilitate all of our WiFi connections.

HRS 02 - INA219 or equivalent voltage sensor shall be used to measure the voltage of the battery pack via I2C with an accuracy of +/- 0.1V

HRS 03 - Adafruit 2941 servo motor or equivalent shall be used to rotate the solar panel fixed onto the gimbal. It shall move 180 degrees laterally and longitudinally.

HRS 04 - L9110S motor driver or equivalent shall be used for driving the DC motor.

HRS 05 - Miniature 3.3V Cooling Fan with Molex PicoBlade Connector for cooling of the battery pack when it overheats.

HRS 06 - Kitronik SOLAR PANEL Model NO: PG-120X62-001 or equivalent solar panel shall be used to charge the battery pack.

HRS 07 - Four photodiodes shall be used to detect the intensity of the light being shined on the solar panel. The angles for the dual axis rotation will be calculated using the irradiance on each of the four photodiodes, two for each axis.

HRS 08 - A 3.7V Li-Ion battery shall be the main battery that will be monitored. It will be used to power the DC motor via a motor driver.

HRS 09 - A 3.7V Li-Ion battery shall be used to power the SAMW25 microcontroller and all sensors.

# Software Requirements Specification (SRS)

SRS 01 - NTC Thermistor to monitor the temperature through BQ2410. We have set the threshold temperature for 45C.

SRS 02 - INA219 or equivalent voltage sensor shall send data over I2C to the microcontroller once per 0.5 seconds +/- 100 milliseconds.

SRS 03 - All sensor data shall be sent to the user interface from the SAMW25 microcontroller using WiFi, updating once per 0.5 seconds +/- 100 milliseconds.

SRS 04 - The light intensity from the photodiodes shall be sent and interpreted by the SAMW25 microcontroller updating every 100 milliseconds +/- 10 milliseconds.

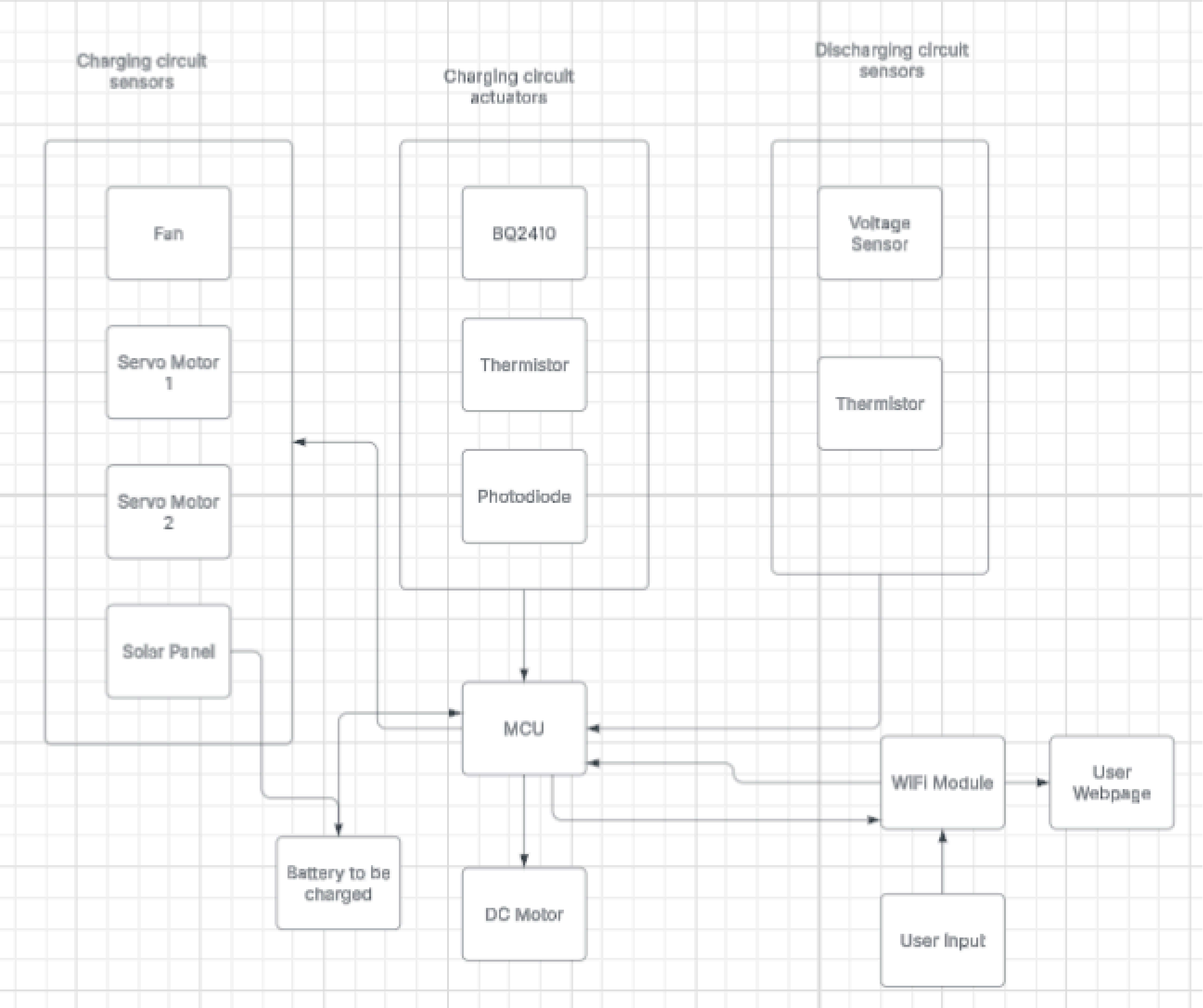
SRS 05 - The light intensity data shall be used in a feedback loop to correct the position of the solar panel toward a higher intensity of light every 100 milliseconds coinciding with the collection of new data.

SRS 06 - L9110S motor driver shall be used for driving the DC motor through user input.

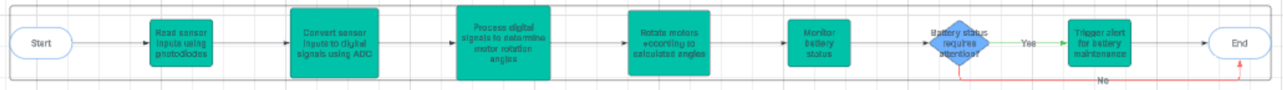
SRS 07 - The voltage, temperature and current state of the battery will be sent to the user and displayed on a webpage for analytics via WiFi module.

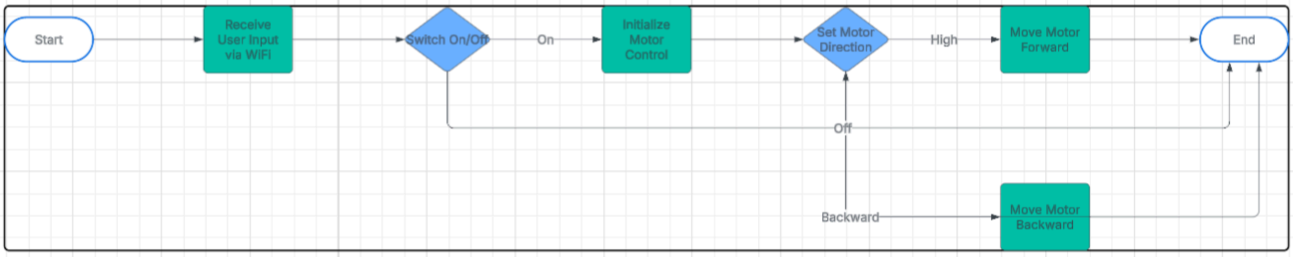
2. The broad system diagram for the 2nd question is attached below.

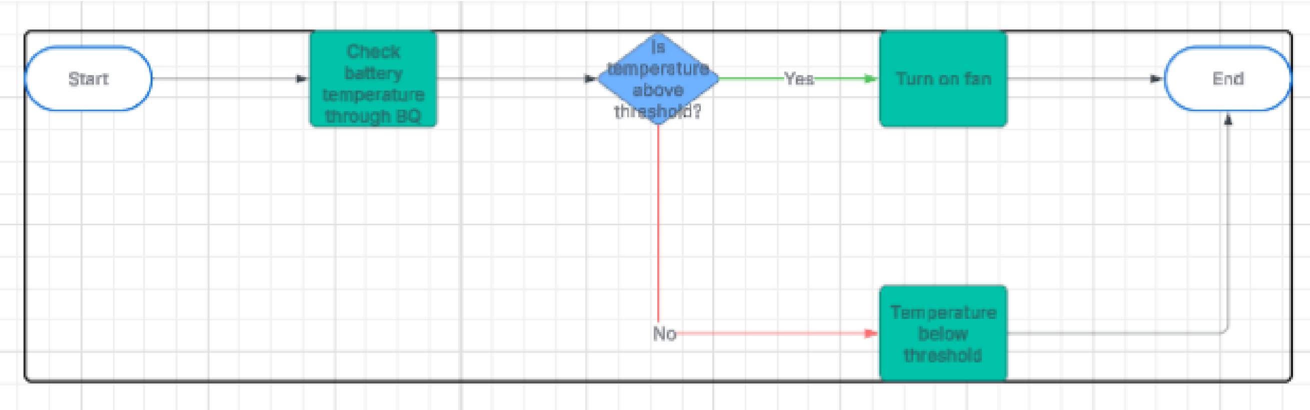
3.

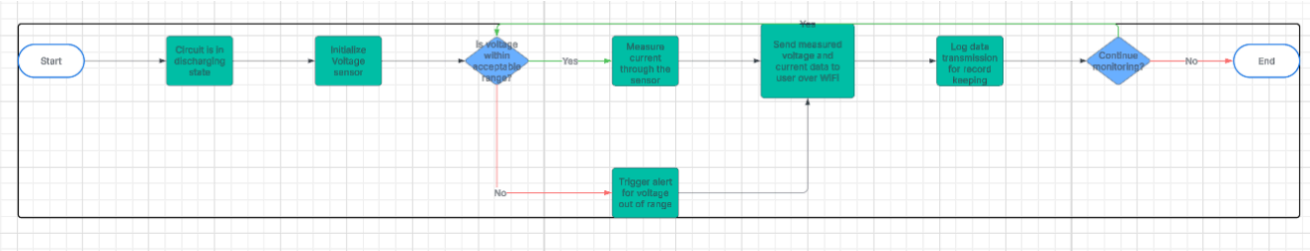


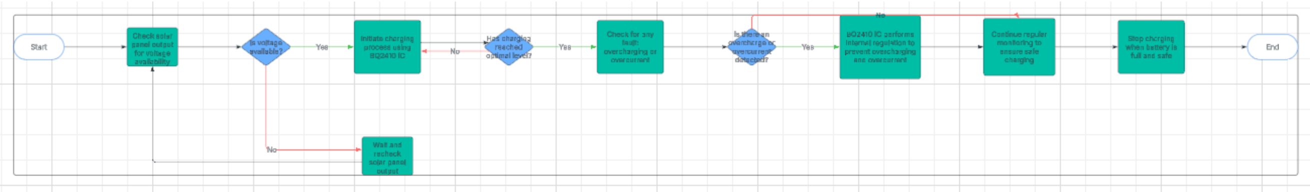
4. I have split the entire system into 5 tasks which I have listed below


Start
Read sensor inputs using photodiodes
Convert sensor inputs to digital signals using ADC
Process digital signals to determine motor rotation angles
Rotate motors according to calculated angles
Monitor battery status
Battery status requires attention?
Yes
Trigger alert for battery maintenance
No
End




Start
Check battery temperature through BQ
Is temperature above threshold?
Yes
Turn on fan
End
No
Temperature below threshold


Start
Circuit is in discharging state
Initialize Voltage sensor
Is voltage within acceptable range?
Yes
Measure current through the sensor
Send measured voltage and current data to user over WIFI
Log data transmission for record keeping
Continue monitoring?
No
End
Yes
No
Trigger alert for voltage out of range


Start
Check solar panel output for voltage availability
Is voltage available?
Yes
Initiate charging process using BQ2410 IC
No
Has charging reached optimal level?
Yes
Check for any fault: overcharging or overcurrent
Is there an overcharge or overcurrent detected?
Yes
BQ2410 IC performs internal regulation to prevent overcharging and overcurrent
No
Continue regular monitoring to ensure safe charging
Stop charging when battery is full and safe
End
No
Wait and recheck solar panel output

# Understanding the Starter Code

1. What does “InitializeSerialConsole()” do? In said function, what is “cbufRx” and “cbufTx”? What type of data structure is it?

   This function initializes the serial communication. It starts by setting the circular buffers Rx and Tx (Rx for receiving data and Tx for transmitting data) equal to initial values. The data structures are circular buffers. It then configures USART and the USART callbacks which includes things like the baud rate of communication.

2. How are “cbufRx” and “cbufTx” initialized? Where is the library that defines them (please list the *C file they come from).

   For intializaton of a circular buffer, you define the head. During a write, the head doesnt change, however, during a read, the head is moved to the next item in the buffer. You then define the tail. In a write, data is placed at the position of the tail and then it moves to the next location. In a read it remains unchanged. It is essentially the opposite of the head. Finally, you define the max, which is the total number of elements the buffer can hold. There is also a boolean which indicates if the buffer is full or not. In the circular_buffer_init() function, first space is allocated in memory with malloc() with the size of the buffer. Then the buffer's data and max size of the buffer is set. The internal state of the buffer is reset with circular_buf_reset() to reset the head and tail.

3. Where are the character arrays where the RX and TX characters are being stored at the end? Please mention their name and size. Tip: Please note cBufRx and cBufTx are structures.

   They are being stored in the structures "cBufRx" and "cBufTx". There are arrays called rxCharacterBuffer[] and txCharacterBuffer[] respectively. the size of each one is 512 characters. These are elements of each structure.

4. Where are the interrupts for UART character received and UART character sent defined?

   They are defined in configure_usart_callbacks(). This function dictates what happens when the "character transmitting" and "character receiving" interrupts have been triggered. It calls "usart_register_callback()" which registers each 'flag' (transmitting or receiving) to callbacks. It then calls usart_enable_callback() which enables these callback functions to happen.

5. What are the callback functions that are called when:

   a. A character is received? (RX)

   usart_read_callback()

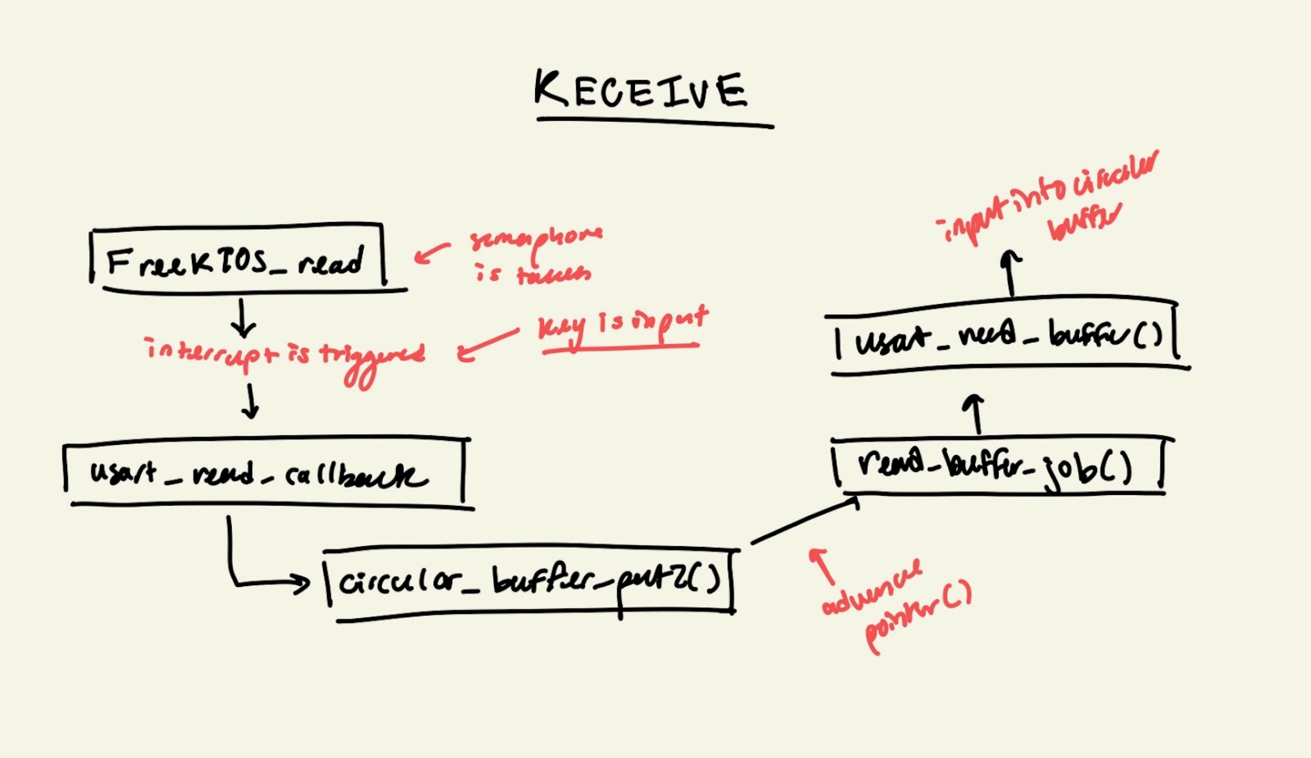
   a. A character has been sent? (TX)

   usart_write_callnack()

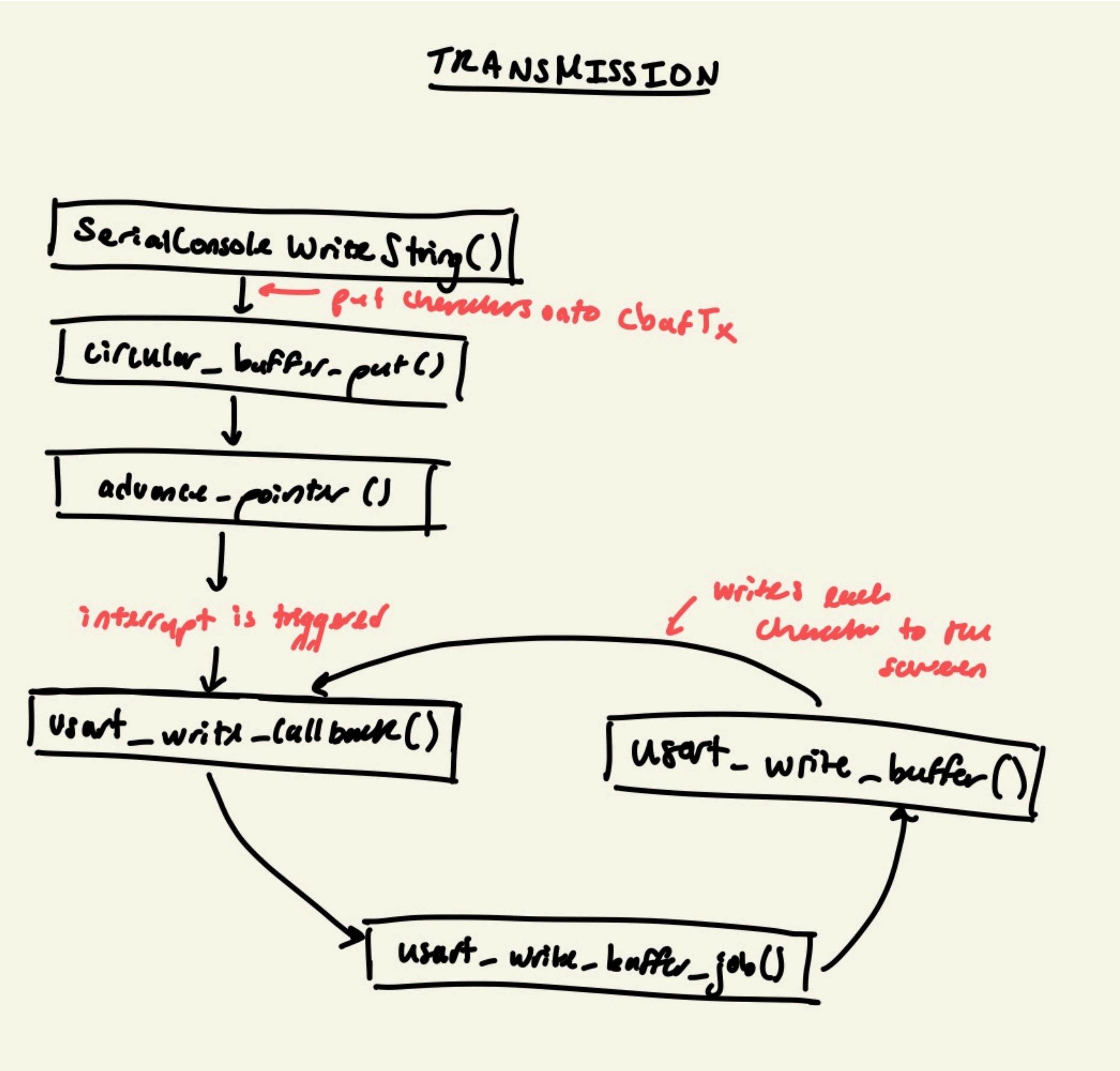
6. Explain what is being done on each of these two callbacks and how they relate to the cbufRx and cbufTx buffers.

   The usart_read_callback() gets called when all of the data has been received that was requested from a UART read and it is ready to receive another byte of data. It would get the data byte from the USART and store it in the buffer cBufRx. The usart_write_callback() gets called when the USART is ready for more data to be sent. This means that it gets the next amount of data that needs to be written from the cBufTx structure. If there is more data to be sent then it sends it to the USART hardware.

7. Draw a diagram that explains the program flow for UART receive – starting with the user typing a character and ending with how that characters ends up in the circular buffer “cbufRx”. Please make reference to specific functions in the starter code.



8. Draw a diagram that explains the program flow for the UART transmission – starting from a string added by the program to the circular buffer “cbufTx” and ending on characters being shown on the screen of a PC (On Teraterm, for example). Please make reference to specific functions in the starter code.

9. 

10. What is done on the function “startStasks()” in main.c? How many threads are started?

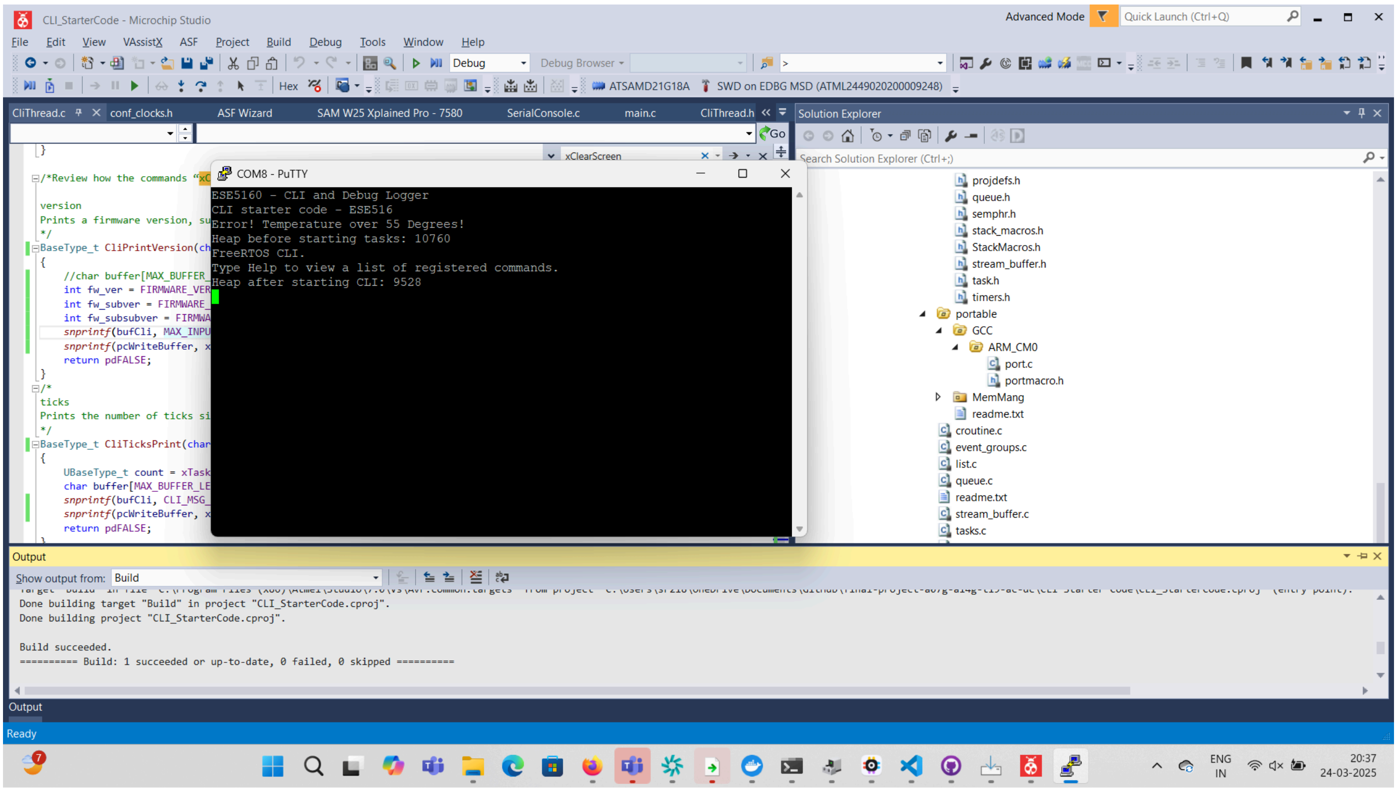
    It initializes tasks needed for the application. It first checks if there is heap size available. Then it initializes all of the tasks needed for the application. Then it creates the CLI task which is represented by vCommandConsoleTask. If it fails, it prints an error message to the serial console. Then it checks the heap memory space after the tasks have been created. There are 2+ threads created. The CLI task is created, it also creates the systme idle task. Finally, depending on how many additional tasks are created, there will be more threads than these intitial 2.

# Debug Logger Module

Please find the code in this file path: final-project-a07g-a14g-t19-ac-dc\A07G_DebugLogger\CLI Starter Code\src\SerialConsole\SerialConsole.c

# Wiretap the Convo

1. What nets must you attach the logic analyzer to? (Check how the firmware sets up the UART in SerialConsole.c!)
2. Where on the circuit board can you attach / solder to?
    1. The logic analyzer can be connected to the SERCOM pins. In our case since we are looking for UART they can be found in PB10 and PB11 which corresponds to SERCOM4.
3. What are critical settings for the logic analyzer?
    1. The logic analyzer needs to know the channels that are input/output, the baud rate for UART, the bit frame- number of data bits, parity bits etc. In our case the baud rate is set to 115200.



```
}

/*Review how the commands "xC

version
Prints a firmware version, su
*/
BaseType_t CliPrintVersion(ch
{
    //char buffer[MAX_BUFFER_
    int fw_ver = FIRMWARE_VER
    int fw_subver = FIRMWARE_
    int fw_subsubver = FIRMWA
    snprintf(bufCli, MAX_INPU
    snprintf(pcWriteBuffer, x
    return pdFALSE;
}
/*
ticks
Prints the number of ticks si
*/
BaseType_t CliTicksPrint(char
{
    UBaseType_t count = xTask
    char buffer[MAX_BUFFER_LE
    snprintf(bufCli, CLI_MSG_
    snprintf(pcWriteBuffer, x
    return pdFALSE;
```

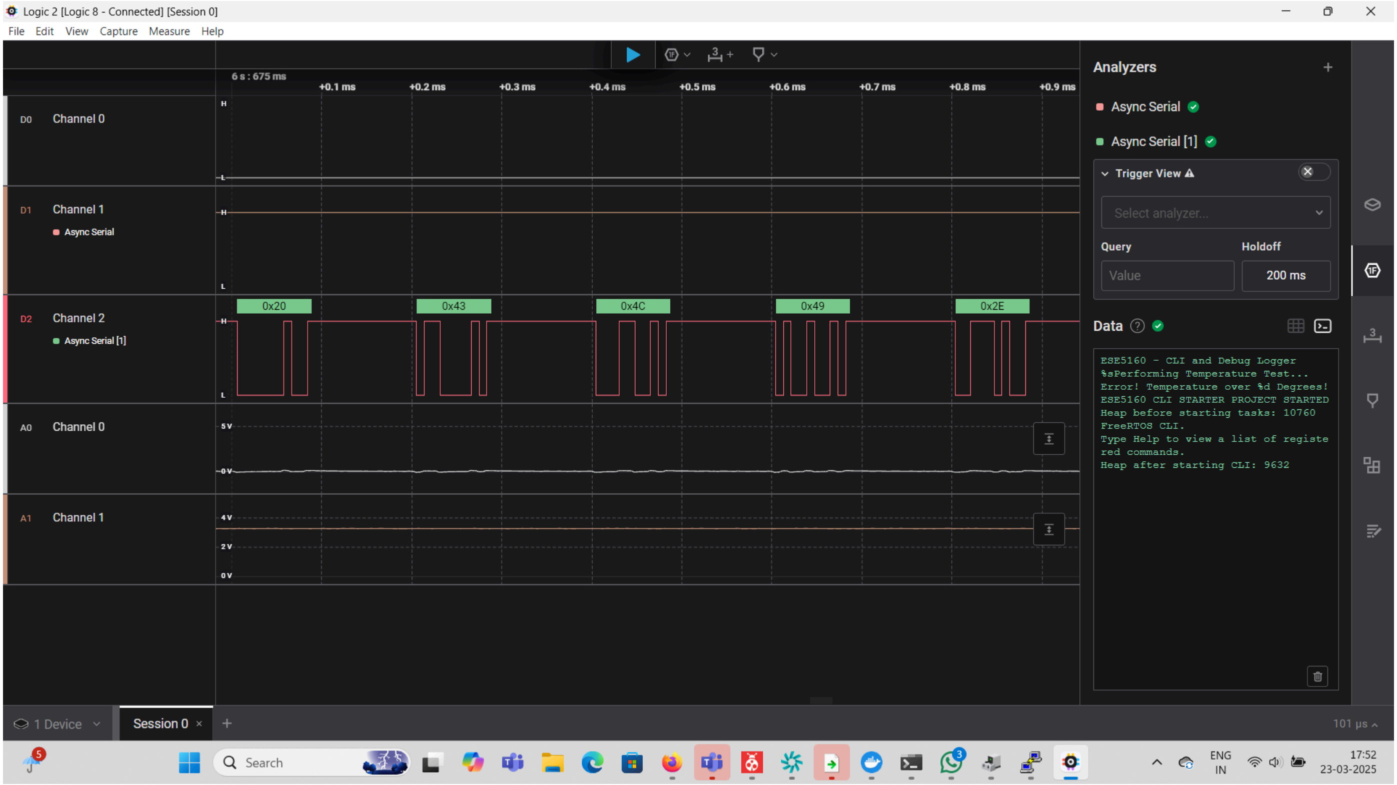
```
ESE5160 - CLI and Debug Logger
CLI starter code - ESE516
Error! Temperature over 55 Degrees!
Heap before starting tasks: 10760
FreeRTOS CLI.
Type Help to view a list of registered commands.
Heap after starting CLI: 9528
```

```
Done building target "Build" in project "CLI_StarterCode.cproj".
Done building project "CLI_StarterCode.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Video link: https://drive.google.com/file/d/1xVz9SYXsYTk-hGG72L9eMZwvI8QJIvLB/view?usp=drive_link

# Logic Analyzer screenshot



```
ESE5160 - CLI and Debug Logger
%sPerforming Temperature Test...
Error! Temperature over %d Degrees!
ESE5160 CLI STARTER PROJECT STARTED
Heap before starting tasks: 10760
FreeRTOS CLI.
Type Help to view a list of registe
red commands.
Heap after starting CLI: 9632
```

The .sal file is linked here:A07G_Part4.sal

The pin connections image is linked here: https://drive.google.com/file/d/1xljPfFQxiu_bJ8iVIRrknJKNnz5zdSSe/view?usp=drive_link

# Add CLI commands

Video link: https://drive.google.com/file/d/1xYLHueic-cgsnsxpwUmjO-qyBjLpzvr9/view?usp=drive_link