a07g-exploring-the-CLI

- Team Number: T05
- Team Name: Fusion Maverick
- Team Members: Qingyang Xu, Ruizhe Wang, Xinyi Wang
- GitHub Repository URL: https://github.com/ese5160/final-project-a07g-a14g-t05-fusion-maverick.git
- Description of test hardware: (development boards, sensors, actuators, laptop + OS, etc)

1. Software Architecture

1. Updated HRS & SRS are elaborated below.

- ***HRS:***
    - *Magic Wand PCBA:*
        - HR01 - project shall be based on SAMW25 core.
        - HR02 - external-connected slide switch shall be used for activation of the wand.
        - HR03 - external-connected Force-Sensitive Resistor(FSR) Interlink Model 402 shall be used for for command detection.
          The sensor will detect if continuous force exerted in the sensing area, as a start flag and maintaining state for the wand gesture recognition.
        - HR04 - mounted 6-axis IMU MPU6500 shall be used for for wand gesture recognition.
          The sensor will collect 6-axis data(3-axis gyroscope and 3-axis accelerometer) while the FSR sensing area is continuously pressed.
          The collected data shall be then used to recognize the swing trajectory of the wand for gesture recognition. The tentative way we've designed for gestures are:
          gesture 1: Wave tilde---wave pattern;
          gesture 2: Zigzag---twinkle;
          gesture 3: Clockwise circle---Turn on the motor;
          gesture 4: Swipe up---Speed up the motor and LCD shows volume up animation;
          gesture 5: Swipe down---Slow down the motor and LCD shows volume down animation;
          gesture 6: Anticlockwise circle---Turn off the motor;
        - HR05 - external-connected LED strip shall be used for command emission indication.
          The LED strip will quickly flash simultaneously as the control command sent out via MQTT to the cloud, imitating the laser emission process.
        - HR06 - external-connceted vibration motor drived by DRV2605L haptic motor controller shall be used for actuator execution feedback.
          The vibration motor will funtion as a feedback reponse to different actuators' actions. The tentative way we're going to execute it is:
            - LCD tasks---Vibration Motor soft fuzz 60%;

        - Motor speed up task---Vibration Motor short double click2-80%;
        - Motor slow down task---Vibration Motor medium click2-80%.

    - *Acuator PCBA:*

        - HR07 - project shall be based on SAMW25 core.

        - HR08 - mounted state LED shall be used to reflect the state of the actuator.
          If there is no control demand, the state LED maintains off; vice versa, the state LED will be turned on when instruction send until the task execution.

        - HR09 - external-connected gearmotor drived by DRV8874 motor driver shall be used as one of the actuator.
          The motor will be drived to execuate the wand command. Such as clockwise to turn on, swipe up to increase the rotation speed, swipe down to decrease the rotation, and anticlockwise to turn off.

        - HR010 - external-connected LCD shall be used as the other actuator.
          The LCD would have two function modes, one is the visulization of motor control, which would reflect the motion state of the motor, such as, as the rotation speed increase, the LCD will display volume up animation.
          In addition, the LCD would solely interact with the magic wand, such as the LCD will antimate a wave pattern when wand has a 'Tilde' tranjectory gesture, and animate a twinkle with respect to wand 'Zigzag' tranjectory gesture.

- ***SRS:***

    - SR01 - slide switch on/off.
        - configured as an external interrupt;
        - programmed state machine controlling the entire state(ON/OFF) of the system.

    - SR02 - FSR based command detection.
        - configured as an external interrupt;
        - a FSR will be used to monitor the use condition of the wand:
            - if no strain/stress detected(***logic 'high'***), the wand is a common stick used for fun;
            - if concontinuous strain/stress detected(***logic 'low'***), the wand is utilized as our proposed "magic wand".

    - SR03 - IMU-based gesture recognition.
        - configued as a SPI(SERCOM0) + one external interrupt;
        - the 6-axis IMU MPU6500 will be used for collecting data when the user touches the sensing area of the strain/stress sensor until no strain/stress detected at that area or stopping swinging of the wand;
          the collected data will be used for the recognition of trajectory gestures of the wand, and then send out the correspoding control demand to the cloud.

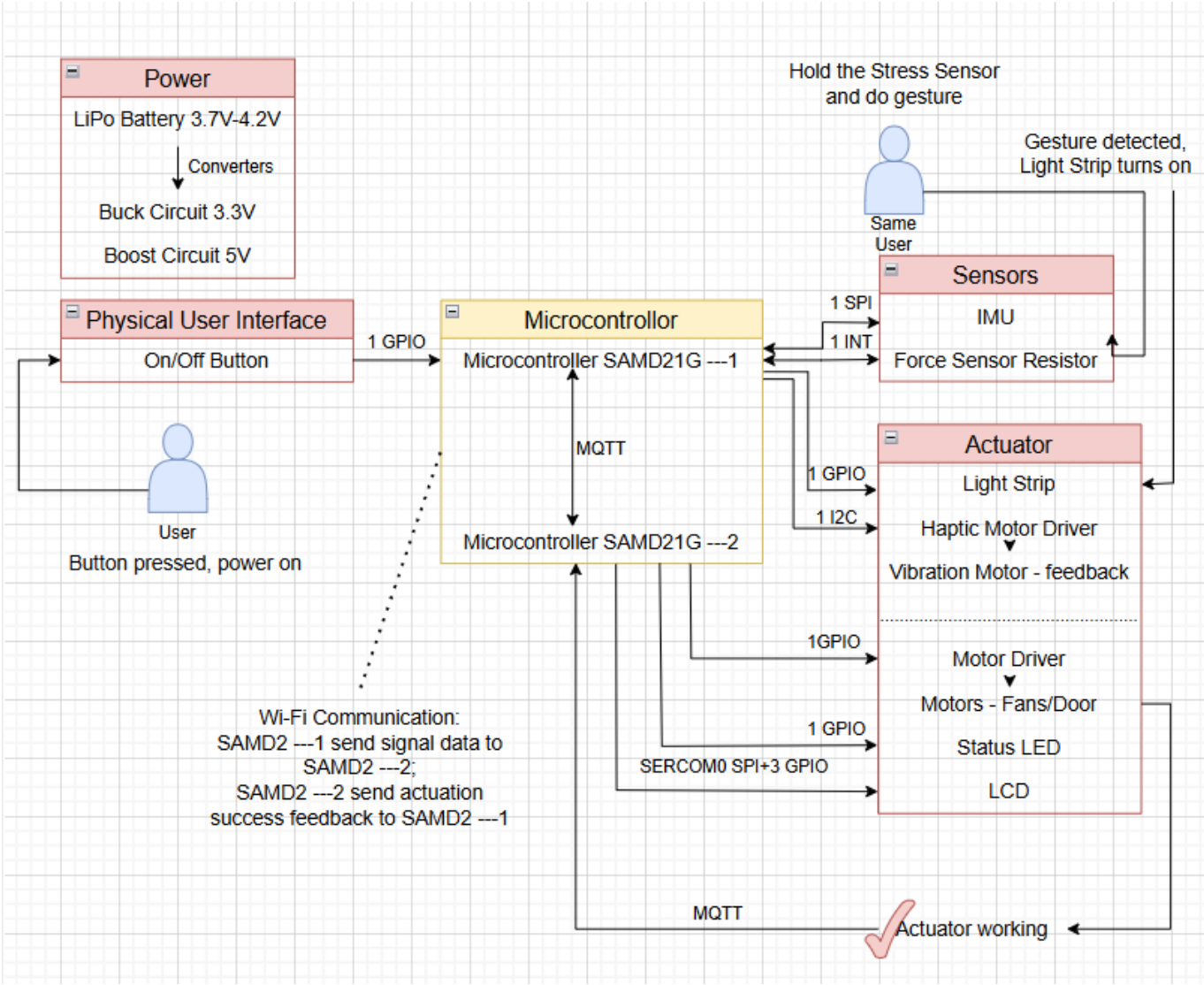
    - SR04 - LED strip based command emission.

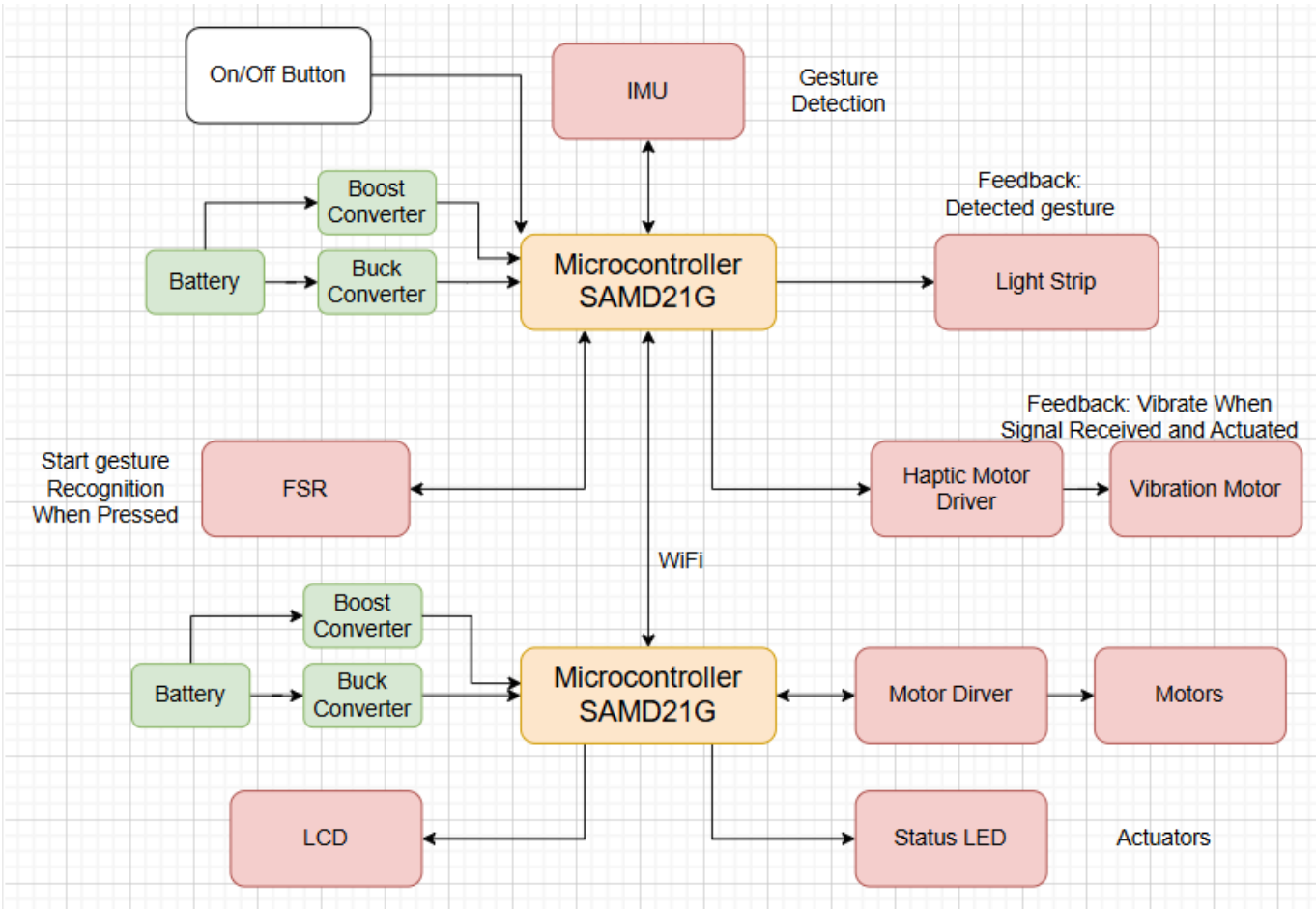
        - configued as a digital output;
        - a LED strip will be programmed to flash simultaneously when the control demand is sent out after correct gesture recogonition.
          Neopixel library will be utilized for this implmentation.

    - SR05 - actuartor execution.
        - the corresponding actuator(determined by the pre-defined gestures) will response to the magic wand instruction, execute the command and send feedback to the cloud to actiavte the vibration motor on the wand;
            - state LED(configued as a a digital output): keep off while no command, turned on when received the command and turned off when the tasks are successfully executed.
            - motor(drived by DRV8874 motor driver, configued as several digital I/Os):
                - activation: clockwise circle drawn by the wand;
                - brake: anticlockwise circle drawn by the wand;
                - accelerate: wand swipes up;
                - decelerate: wand swipes down.
            - LCD(configued as a SPI(SERCOM0) + several digital I/Os):
                - mode 1: visulization of motor motion state:
                    - motor accelerates: volume up animation;
                    - motor decelerates: volume down animation.
                - mode 2: intaction with wand:
                  the LCD would solely interact with the wand. For instance, the LCD will animate a twinkle with respect to wand 'Zigzag' tranjectory gesture; the LCD will angimate a wave pattern with respect to wand Tilde' tranjectory gesture.

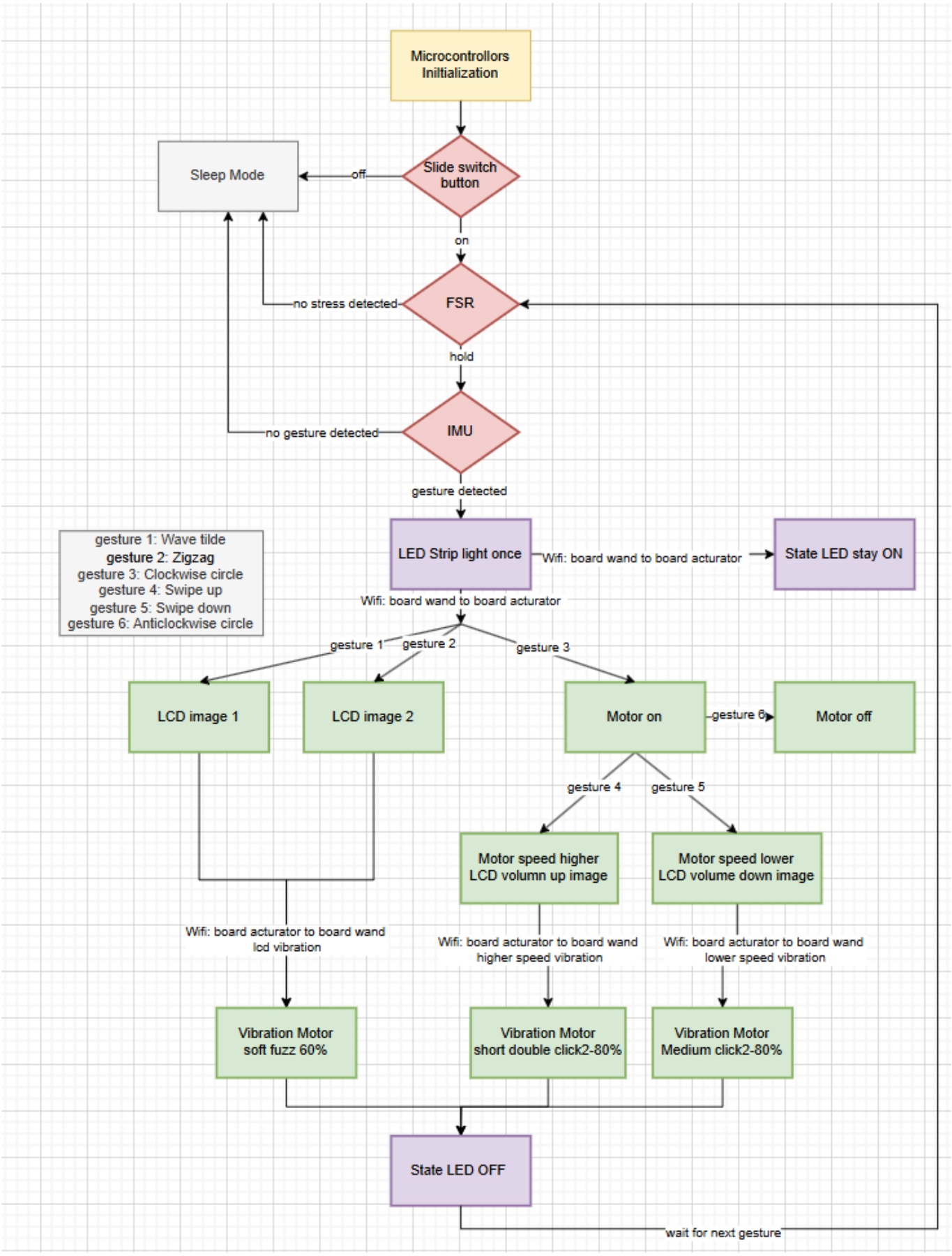
    - SR06 - vibration motor based actuator exection feedback.
        - drived by DRV2605L haptic motor controller, configured as an I2C(SERCOM3) + one digital output
        - a vibration motor will be activated for about few seconds once the control demand has been successfully received and executed by the actuator. We proposed varying vibration effects/modes of the motor with respect to diffrent task accomplishment. The tentative way we're going to execute it is:
            - LCD tasks---Vibration Motor soft fuzz 60%;
            - Motor speed up task---Vibration Motor short double click2-80%;
            - Motor slow down task---Vibration Motor medium click2-80%;

2. Block diagrams are shown below.


Power
LiPo Battery 3.7V-4.2V
Converters
Buck Circuit 3.3V
Boost Circuit 5V
Hold the Stress Sensor and do gesture
Same User
Gesture detected, Light Strip turns on
Sensors
IMU
Force Sensor Resistor
1 SPI
1 INT
Physical User Interface
On/Off Button
1 GPIO
Microcontrollor
Microcontroller SAMD21G ---1
MQTT
Microcontroller SAMD21G ---2
Actuator
Light Strip
Haptic Motor Driver
Vibration Motor - feedback
1 GPIO
1 I2C
1GPIO
Motor Driver
Motors - Fans/Door
Status LED
LCD
1 GPIO
SERCOM0 SPI+3 GPIO
User
Button pressed, power on
Wi-Fi Communication:
SAMD2 ---1 send signal data to SAMD2 ---2;
SAMD2 ---2 send actuation success feedback to SAMD2 ---1
MQTT
Actuator working


On/Off Button
IMU
Gesture Detection
Boost Converter
Battery
Buck Converter
Microcontroller SAMD21G
Feedback: Detected gesture
Light Strip
Feedback: Vibrate When Signal Received and Actuated
Start gesture Recognition When Pressed
FSR
Haptic Motor Driver
Vibration Motor
WiFi
Boost Converter
Battery
Buck Converter
Microcontroller SAMD21G
Motor Dirver
Motors
LCD
Status LED
Actuators

3. Flowchart illusatration is shown below.



2. Understanding the Starter Code

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

- The function `InitializeSerialConsole()` initializes the UART for serial communication by setting up circular buffers for receiving (`cbufRx`) and transmitting (`cbufTx`) data, configuring the USART module and its callbacks, setting interrupt priority, and initiating a read operation to enable continuous data reception. The circular buffers are created using `circular_buf_init()`, which allocates memory for storing serial data before it is processed.

- `cbufRx` and `cbufTx` are in circular buffer data structure, handle data type, a type of FIFO (First-In, First-Out) data structure that allows efficient handling of continuous data streams. This structure enables smooth serial communication by storing incoming and outgoing data in a fixed-size buffer where the read and write pointers wrap around when they reach the buffer's end. Circular buffers help prevent data loss, making them ideal for asynchronous serial data handling in embedded systems.

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

- The variables `cbufRx` and `cbufTx` are initialized in the function `InitializeSerialConsole()` using the function `circular_buf_init()`. This function is called with two parameters: a pointer to a character buffer(`rxCharacterBuffer` for `cbufRx` and `txCharacterBuffer` for `cbufTx`) and the respective buffer sizes(`RX_BUFFER_SIZE` and `TX_BUFFER_SIZE`). This setup effectively creates circular buffers for handling received and transmitted data in the serial communication process.

- The `circular_buf_init()` function is defined in `circular_buffer.c` implementation file, with its corresponding declarations in `circular_buffer.h` header file. This library implements a circular buffer(ring buffer) structure, enabling efficient handling of serial data by maintaining a FIFO (First-In, First-Out) queue.
  The `SerialConsole.c` implementation file, which includes the `InitializeSerialConsole()` function, utilizing this library to initialize and manage the serial console's buffers.

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.

- `rxCharacterBuffer` – This array stores the received characters (RX).
  - Size: `RX_BUFFER_SIZE`(512 bytes).
  - Defined in: `SerialConsole.c` implementation file.

- `txCharacterBuffer` – This array stores the transmitted characters (TX).
  - Size: `TX_BUFFER_SIZE`(512 bytes).

- Defined in: `SerialConsole.c` implementation file.

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

- These two interrupts are defined within the `configure_usart_callbacks()` function in `SerialConsole.c` implementation file, which declares and enables the interrupt service routine(ISR) - callback functions of these two operations within the USART module.

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

- ***a. A character is received? (RX)***
  - `usart_read_callback()`.
- ***b. A character has been sent? (TX)***
  - `usart_write_callback()`.

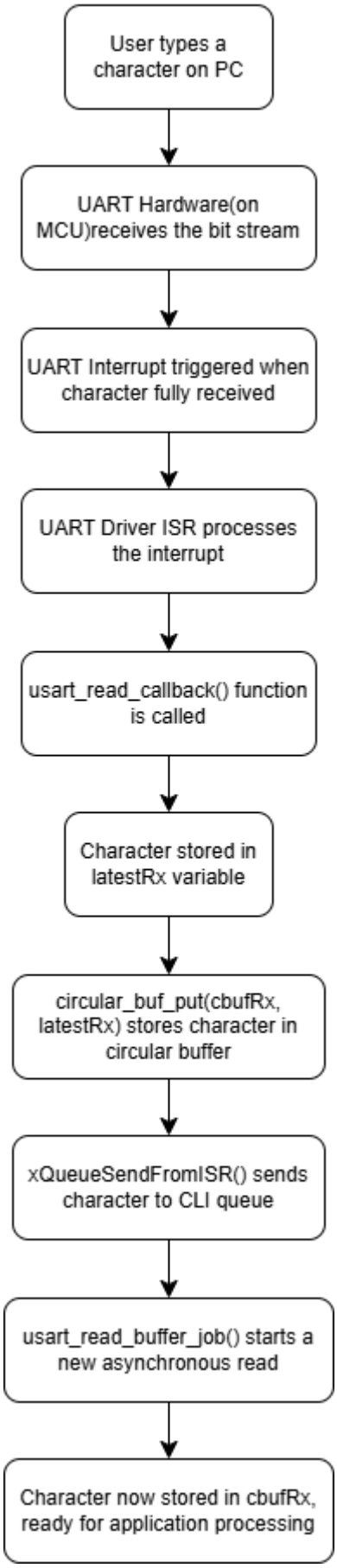
6. Explain what is being done on each of these two callbacks and how they relate to the cbufRx and cbufTx buffers.

- `usart_read_callback()`:
  - Stores the received character(from `latestRx`) into the circular receive buffer `cbufRx` if `cbufRx` is not full;
  - Sends the character to the CLI queue to notify the CLI thread about the new input;
  - Starts a new asynchronous read operation to receive the next character;
  - Yields to a higher priority task if one was woken by the queue operation.
- `usart_write_callback()`:
  - Attempts to get the next character to transmit from the circular transmit buffer `cbufTx`;
  - If a character is available(`cbufTx` is not empty), it starts a new asynchronous write operation to send that character;
  - If no character is available(`cbufTx` is empty), it simply returns, ending the transmission chain until new data is added to the buffer `cbufTx`.

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.

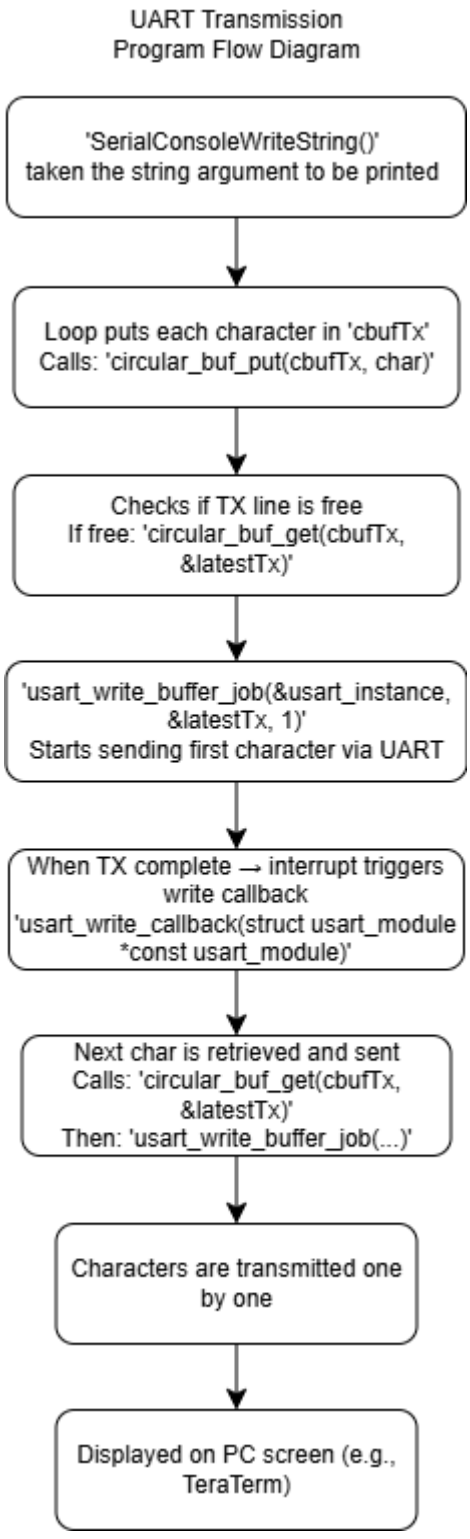
- Seen the UART receive progress mapping diagram below.

UART Receive Program
Flow Diagram



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.

- Seen the UART transmission progress mapping diagram below.



9. What is done on the function "startStasks()" in main.c? How many threads are started?

- According to the function body of `StartTasks()`, it execute the following actions:

1. Prints the available heap memory before starting any tasks;
2. Creates the Command-Line Interface (CLI) task using `xTaskCreate(vCommandConsoleTask, "CLI_TASK", CLI_TASK_SIZE, NULL, CLI_PRIORITY, &cliTaskHandle)`;
    - Prints an error message if the CLI task creation fails;
3. Displays the remaining heap memory after starting the CLI task.

- Only 1 thread is started in this function - the CLI task that handles the command-line interface functionality.

3. Debug Logger Module

Seen the updated Debug Logger Module in GitHub repository or the below code snippet.

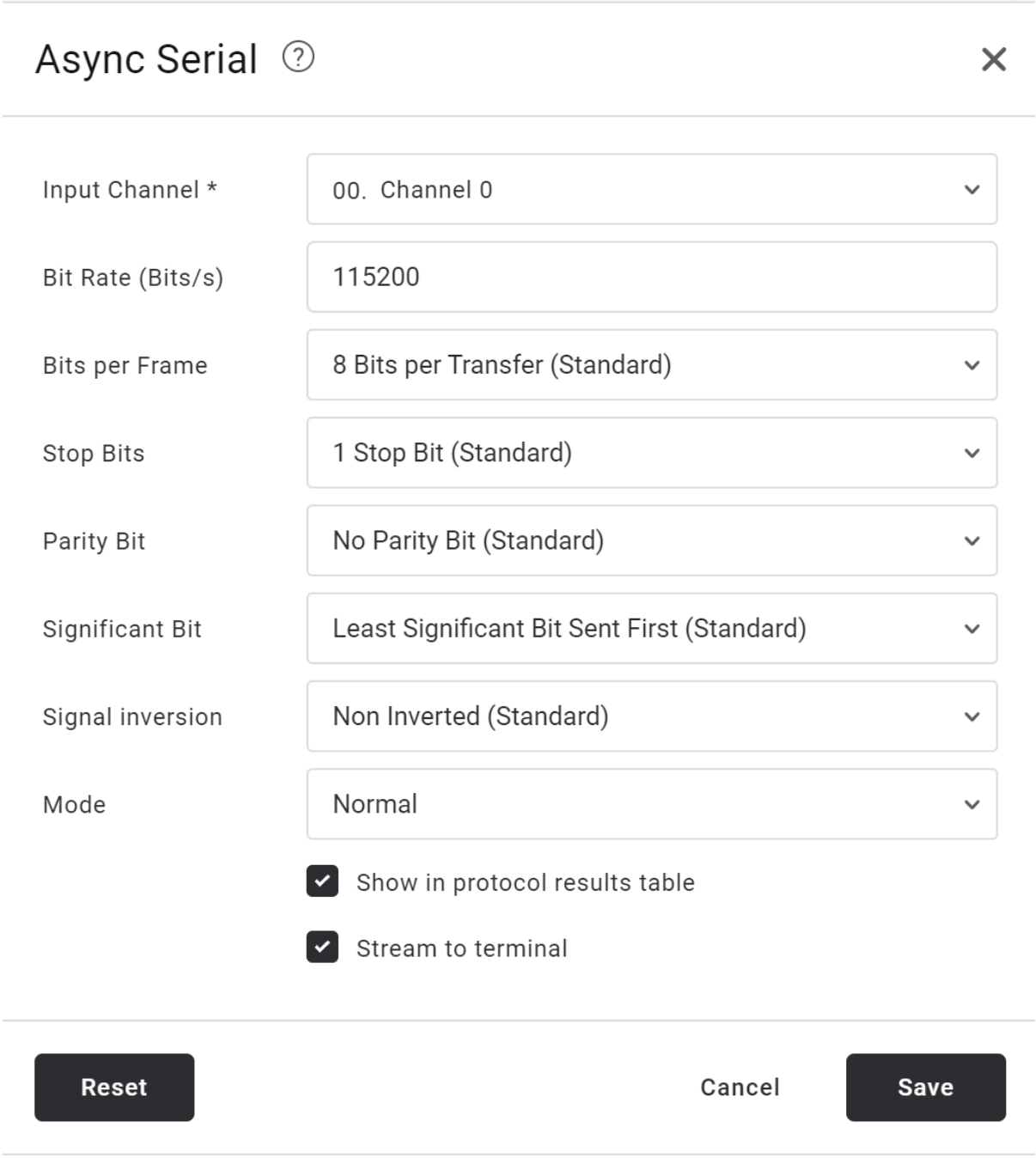
```
// Header file
/**
 * @fn          LogMessage
 * @brief       Logs a message at the specified debug level.
 * @param       level  Determines the log levels of the message to output. If the
level is smaller than
 *                     the current “logLevel” it is not printed.
 * @param       format Pointer to a array of characters to be printed.
 * @param       ...    Optional variables used in the format string. These are
handled
 *                     using a variable argument list (va_list). In our project,
it shall be sensor reading and states indicating, etc.
 * @note            Use vsprintf() to format the message and send to the UART
terminal.
 ******************************************************************************/
void LogMessage(enum eDebugLogLevels level, const char *format, ...);


// Implementation files
/**
 * @brief Logs a message at the specified debug level.
 */
void LogMessage(enum eDebugLogLevels level, const char *format, ...)
{
    // Check if the message should be printed based on current log level
    if (level < getLogLevel())
    return;

    // Prepare buffer for formatted message
    char logBuffer[256];  // Make sure this is big enough for your messages
    va_list args;
    va_start(args, format);
    vsprintf(logBuffer, format, args);  // UNSAFE if logBuffer is too small
    va_end(args);

    // Send formatted string to UART
```

```
    SerialConsoleWriteString(logBuffer);
}
```

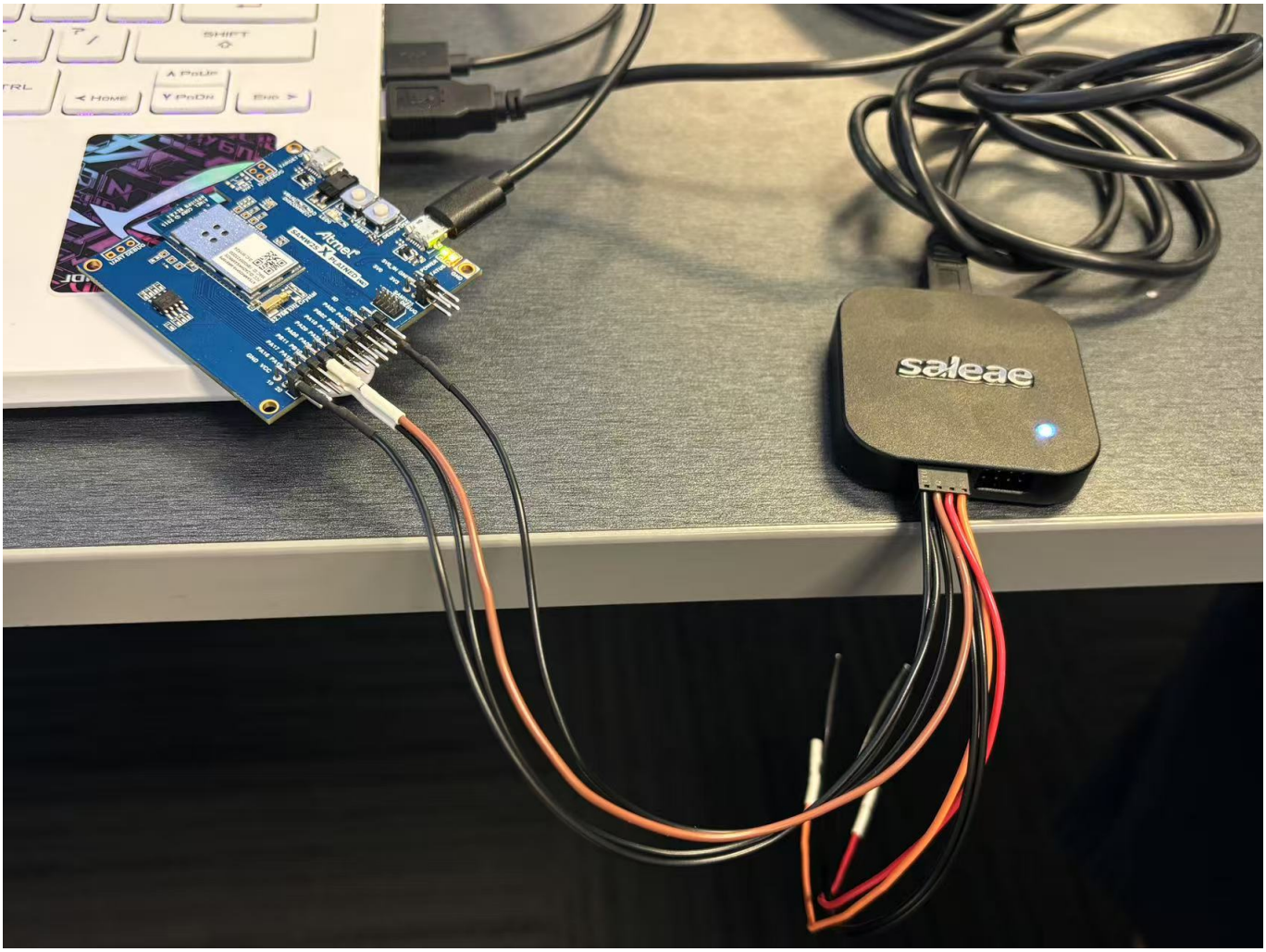
4. Wiretap the convo!

1. Quick Questions

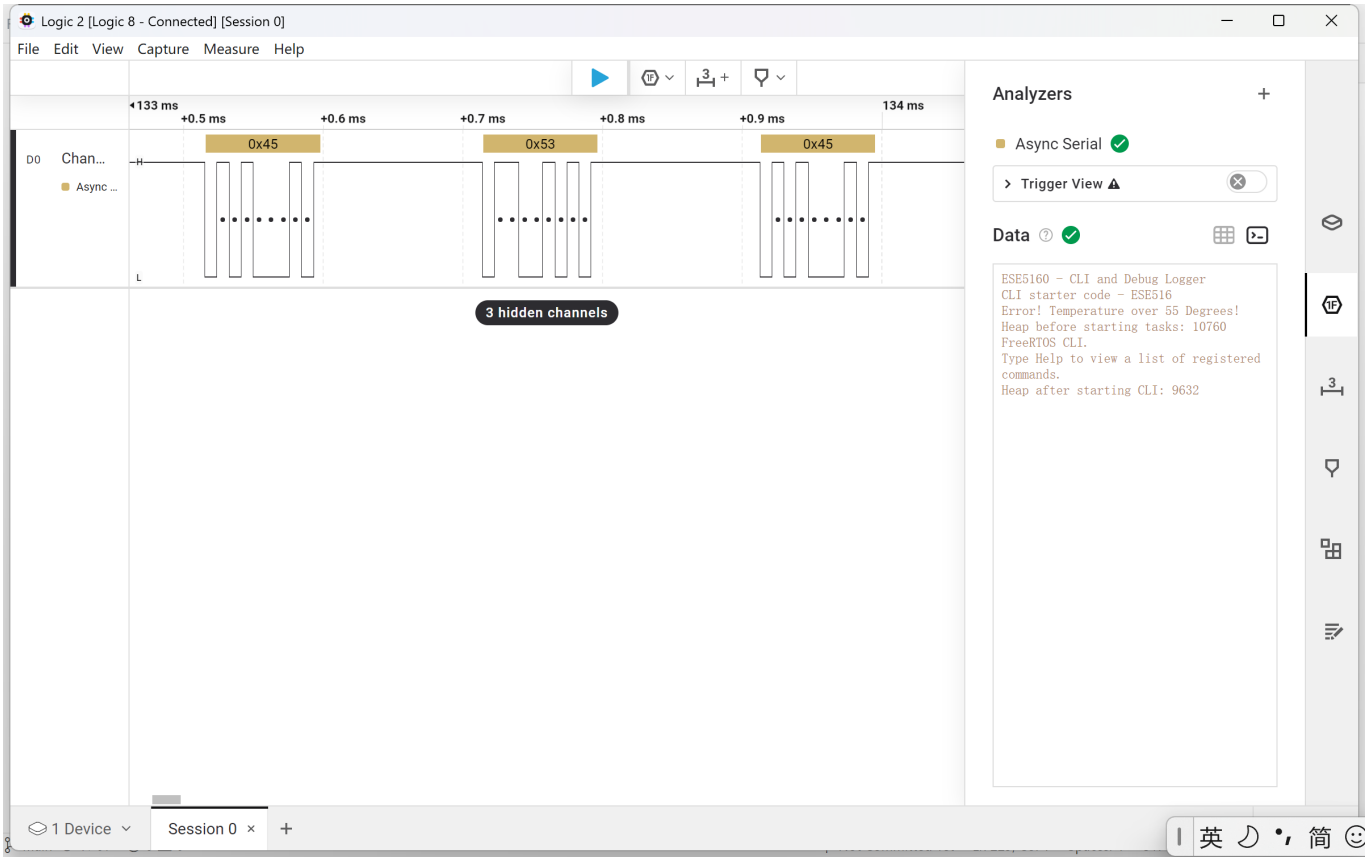
- ***1. What nets must you attach the logic analyzer to? (Check how the firmware sets up the UART in SerialConsole.c!)***
    - PB10 is used as SERCOM4 PAD[2], which is UART TX from the SAMW25 to the EDBG.
    - GND is used as ground reference.
- ***2. Where on the circuit board can you attach / solder to?***
    - There is a header for PB10 on the dev board. We connect PB10 with chanel 0 signal for the logic analyzer, and GND connected to the chanel 0 gnd for the logic analyzer.
- ***3. What are critical settings for the logic analyzer?***
    - Critical settings are as follows, enuring the configuration of the logic analyzer are the same as dev board UART setting.


Async Serial
Input Channel *
00. Channel 0
Bit Rate (Bits/s)
115200
Bits per Frame
8 Bits per Transfer (Standard)
Stop Bits
1 Stop Bit (Standard)
Parity Bit
No Parity Bit (Standard)
Significant Bit
Least Significant Bit Sent First (Standard)
Signal inversion
Non Inverted (Standard)
Mode
Normal
Show in protocol results table
Stream to terminal
Reset
Cancel
Save

2. Hardware connections



3. Decoded message



4. sal file Sal file link.

5. Complete the CLI

See code below CliThread.c and SerialConsole.c.

6. Add CLI commands

See code below CliThread.c and CliThread.h

Demonstration video: Video