Internet of Things lab 2 Fall 2023

Avhishek Biswas

Task 1 :

Requirements

1. **Blue LED**: The Blue LED should toggle its state every 0.5 seconds.
2. **Yellow LED**: The Yellow LED should toggle its state every 1 second.
3. **Serial Monitor Output**: The program should output the current state of each LED ("on" or "off") to the Serial Monitor whenever the LED state changes.

Development Plan:

a. Procedure of Solving the Problem

1. **Initialization**: Open the Arduino IDE, select the SparkFun RF Pro board, and the appropriate port.
2. **Pin Configuration**: Use `pinMode()` to set the digital pins for the Blue and Yellow LEDs as output.
3. **Timing Control**: Use the `millis()` function to keep track of the time elapsed since the board started running.
4. **LED Toggling**: Use conditional statements to toggle the LEDs based on the time intervals specified.
5. **Serial Output**: Use `SerialUSB.println()` to output the current state of each LED to the Serial Monitor.
6. **Testing and Debugging**: Upload the code to the SparkFun RF Pro board and test the functionality. Debug if necessary.

b. Configuration Table

Requirement	Register Name	Register Function	Register Value
Blue LED Pin	N/A	Digital Pin Configuration	`PIN_LED_13`

Requirement	Register Name	Register Function	Register Value
Yellow LED Pin	N/A	Digital Pin Configuration	`PIN_LED_RXL`
Serial Monitor Setup	N/A	Serial Communication	`9600` baud

2. Development Process:

1. **Initialization**:
   - Open the Arduino IDE and select the SparkFun RF Pro board from the board manager.
   - Select the appropriate port to which the board is connected.
2. **Pin Configuration**:
   - Use the `pinMode()` function to configure the digital pins for the Blue and Yellow LEDs (`PIN_LED_13` and `PIN_LED_RXL`) as output pins.
3. **Serial Communication Setup**:
   - Initialize the Serial communication with a baud rate of 9600 using `SerialUSB.begin(9600)` to enable debugging and monitoring.
4. **Variable Initialization**:
   - Initialize variables `previousMillisBlue` and `previousMillisYellow` to store the last time each LED was toggled.
   - Initialize `intervalBlue` and `intervalYellow` to set the time intervals for toggling the Blue and Yellow LEDs, respectively.
5. **Timing Control**:
   - Use the `millis()` function to get the current time in milliseconds since the board started running.
   - Store this value in a variable called `currentMillis` .
6. **LED Toggling**:
   - Use conditional statements to compare `currentMillis` with `previousMillisBlue` and `previousMillisYellow` .
   - If the difference exceeds `intervalBlue` or `intervalYellow` , toggle the respective LED and update `previousMillisBlue` or `previousMillisYellow` with `currentMillis` .
7. **Serial Output**:
   - Use `SerialUSB.println()` to output the current state of each LED to the Serial Monitor. This helps in debugging and provides a real-time status of the LEDs.
8. **Testing and Debugging**:
   - Upload the code to the SparkFun RF Pro board and open the Serial Monitor to observe the LED states.

- Make any necessary adjustments to the code and re-upload if needed.

9. **Final Verification**:
    - Confirm that the LEDs are toggling at the expected intervals and that the Serial Monitor output is correct.

Approach 1: Multi-tasking using millis()

```
// LED definitions in the datasheet
// D13 (PIN_LED_13): Blue
// RX (PIN_LED_RXL): Yellow

// Variables will change:
long previousMillisBlue = 0;  // will store the last time Blue LED was updated
long previousMillisYellow = 0;  // will store the last time Yellow LED was updated

// intervals at which to blink (milliseconds)
long intervalBlue = 500;
long intervalYellow = 1000;

void setup() {
  // set the digital pins as output:
  pinMode(PIN_LED_13, OUTPUT);
  pinMode(PIN_LED_RXL, OUTPUT);
  SerialUSB.begin(9600);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.
  unsigned long currentMillis = millis();

  // Check if it's time to toggle the Blue LED
  if (currentMillis - previousMillisBlue > intervalBlue) {
    previousMillisBlue = currentMillis;
    if (digitalRead(PIN_LED_13) == LOW) {
      digitalWrite(PIN_LED_13, HIGH);
      SerialUSB.println("Blue LED is on");
    } else {
      digitalWrite(PIN_LED_13, LOW);
      SerialUSB.println("Blue LED is off");
    }
  }

  // Check if it's time to toggle the Yellow LED
  if (currentMillis - previousMillisYellow > intervalYellow) {
    previousMillisYellow = currentMillis;
    if (digitalRead(PIN_LED_RXL) == LOW) {
      digitalWrite(PIN_LED_RXL, HIGH);
      SerialUSB.println("Yellow LED is on");
    } else {
      digitalWrite(PIN_LED_RXL, LOW);
      SerialUSB.println("Yellow LED is off");
    }
  }
}
```

3. Test Plan:

1. **LED Blinking**:
    - Verify that the Blue LED toggles on and off at an interval of 500 milliseconds.
    - Verify that the Yellow LED toggles on and off at an interval of 1000 milliseconds.
2. **Serial Output**:
    - Verify that the Serial Monitor displays the correct LED status messages ("Blue LED is on/off", "Yellow LED is on/off") in real-time.

Component	Test Description	Result	Comment
Blue LED	Toggle on/off at 1000ms interval	Pass	LED toggled as expected at 1-second intervals
Blue LED	Toggle on/off at 500ms interval	Pass	LED toggled as expected at 0.5-second intervals
Yellow LED	Toggle on/off at 1000ms interval	Pass	LED toggled as expected at 1-second intervals
Serial Output	Display "Blue LED is on/off"	Pass	Messages displayed correctly in the serial monitor
Serial Output	Display "Yellow LED is on/off"	Pass	Messages displayed correctly in the serial monitor
System Level	Both LEDs toggle at correct intervals	Pass	Both LEDs toggled at their respective intervals without conflict

Output:

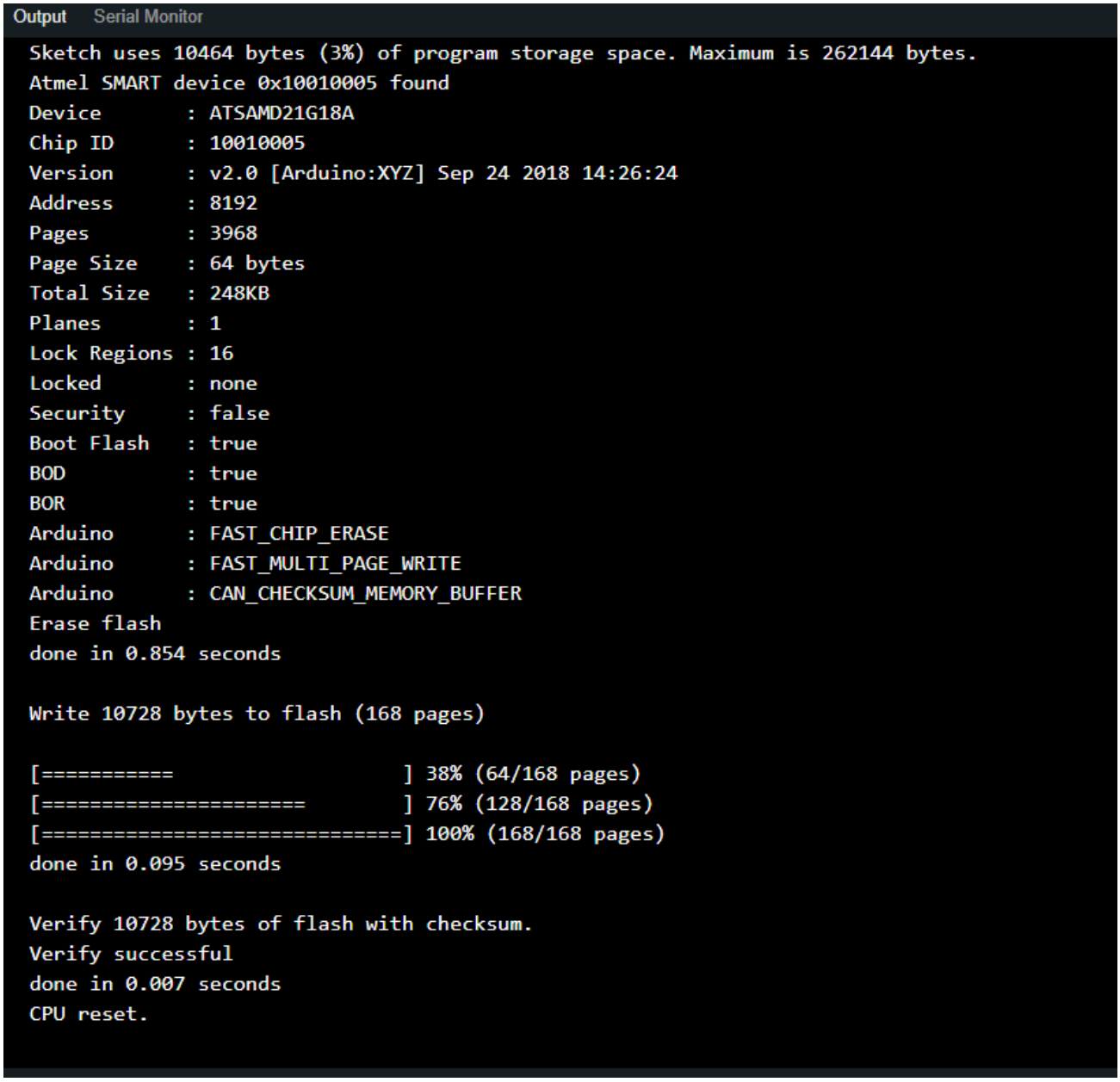

Fig.1 - Console Output.

Screenshot

Fig.2 - Output in the serial monitor.

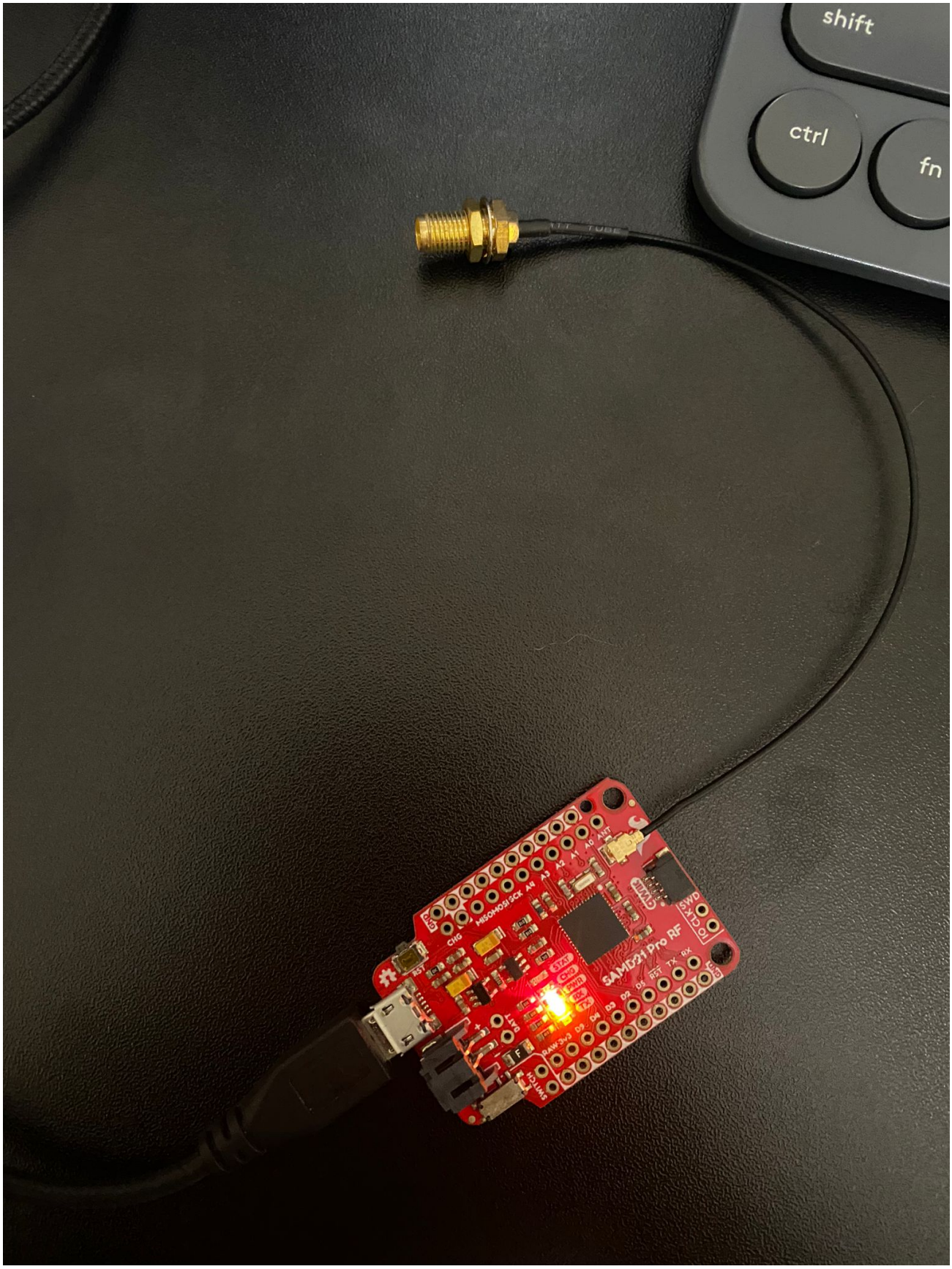

Fig.3 - Only blinking of Yellow LED.

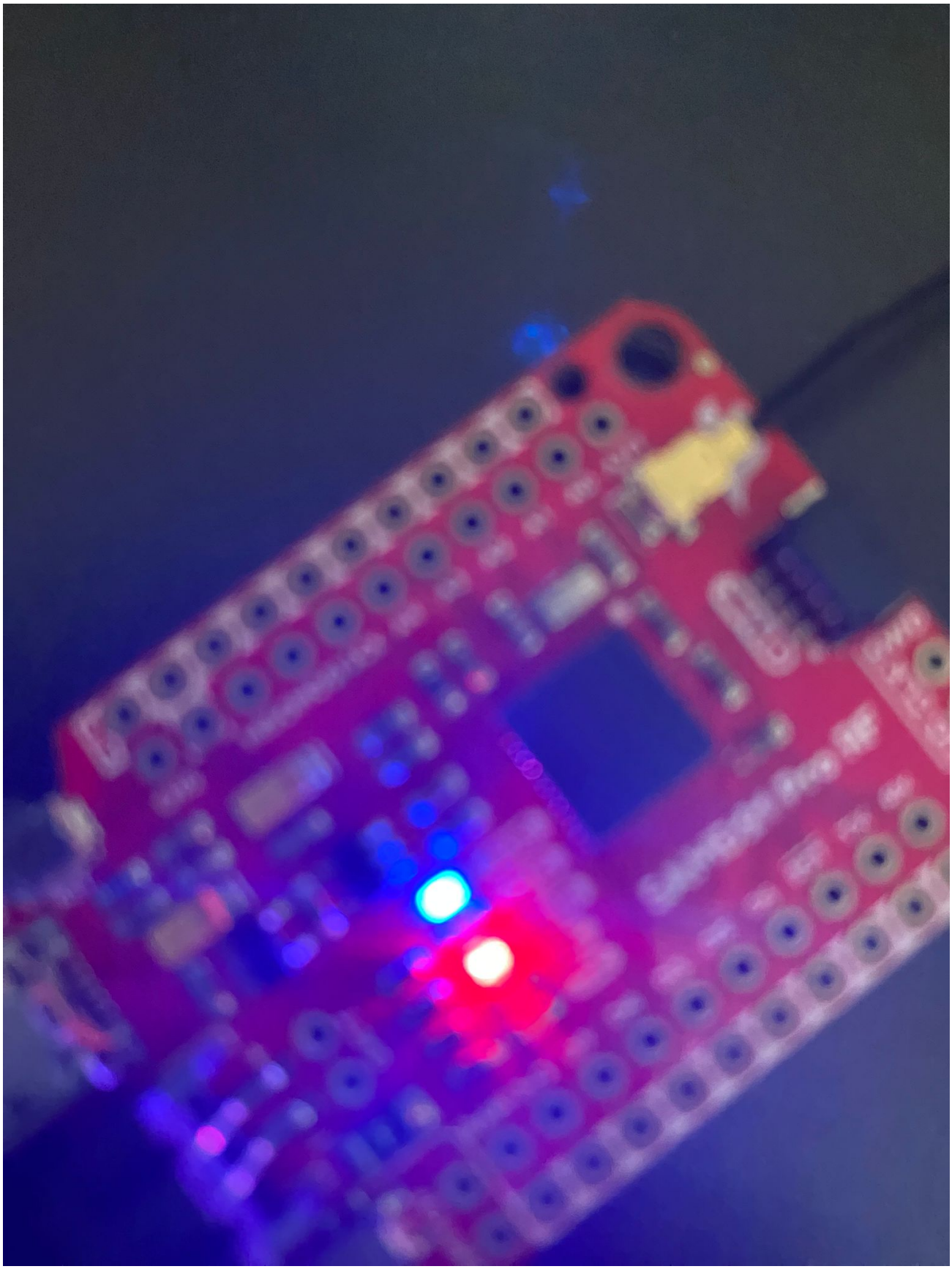

Fig.4 - Only blinking of Blue LED.

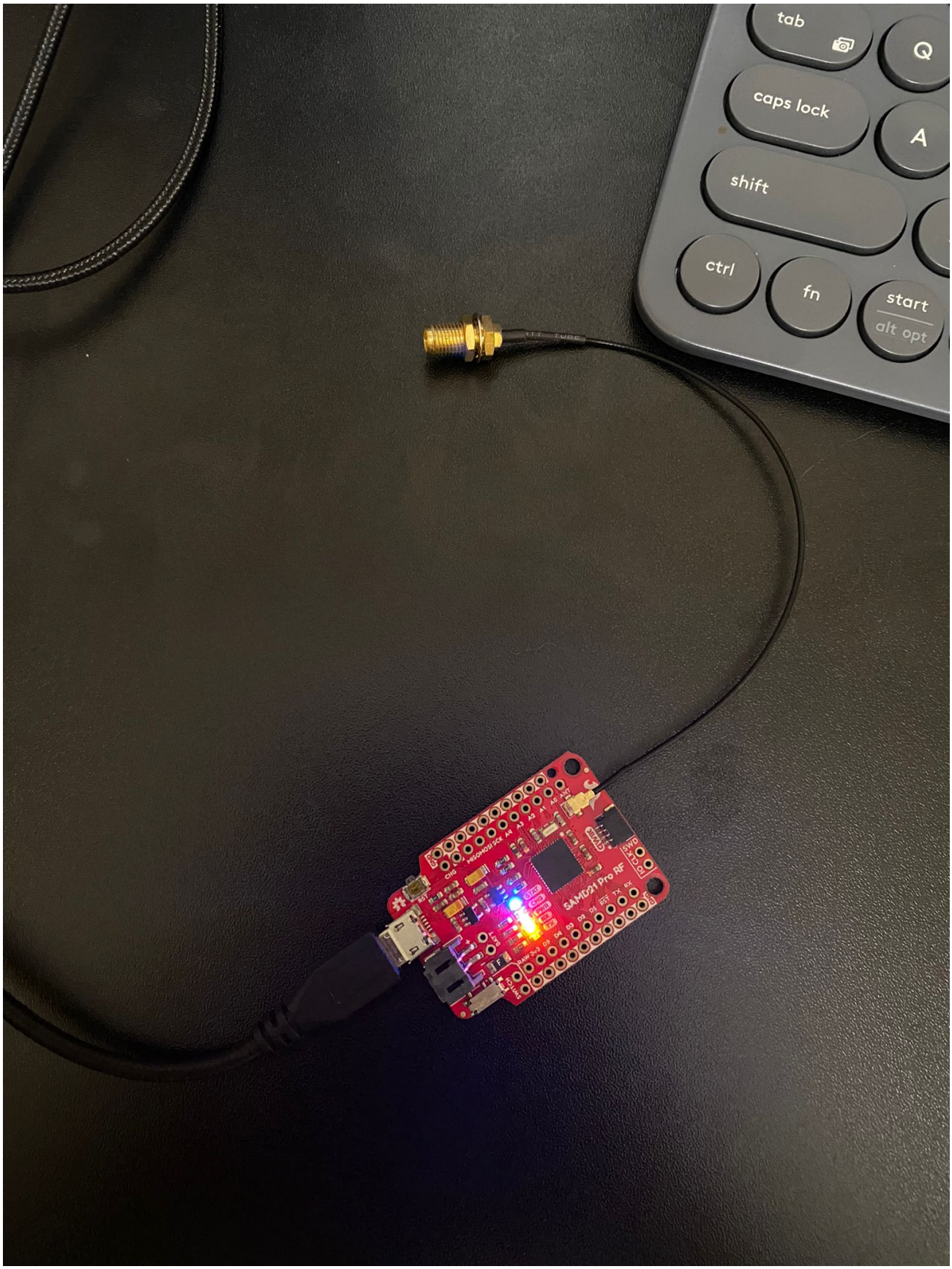

Fig.5 - Blinking of both Yellow and Blue LED.

Video Link

See working video - Link to Video.

Task 2 :

Requirements

1. **Blue LED**: The Blue LED should toggle its state every 0.5 seconds.
2. **Yellow LED**: The Yellow LED should toggle its state every 1 second.
3. **Serial Monitor Output**: The program should output the current state of each LED ("on" or "off") to the Serial Monitor whenever the LED state changes.

Development Plan

a. Procedure of Solving the Problem

1. **Requirements Analysis**: The first step was to understand the requirements. We needed to toggle two LEDs at different frequencies using two different timer interrupts (TC3 and TC4) on a SAMD21 board.
2. **Design Phase**:
   - Decided to use the 16-bit timers TC3 and TC4 in Match Frequency Operation mode.
   - Calculated the compare values based on the CPU frequency and the desired LED toggle frequency.
   - Designed the interrupt handlers for each timer.
3. **Implementation Phase**:
   - Configured the timers in the `startTimer()` function.

- Implemented the interrupt handlers `TC3_Handler()` and `TC4_Handler()` to toggle the LEDs and print the LED states to the Serial monitor.

4. **Testing Phase**:
   - Unit tests were performed to ensure each LED toggled at the correct frequency.
   - System-level tests were performed to ensure that both LEDs could toggle simultaneously without issues.
5. **Debugging and Optimization**:
   - Combined lines in `startTimer()` to reduce the function length as per the task requirement.
   - Ensured that the overflow interrupt (OVF) was used instead of the match interrupt (MC0).
6. **Documentation**:
   - Added inline comments to the code for better readability and maintenance.
   - Created this development plan to outline the procedure and configurations used.

b. Configuration Table

Register name	Register function	Register value
`TC->CTRLA`	Timer Control A	`TC_CTRLA_MODE_COUNT16`, `TC_CTRLA_WAVEGEN_MFRQ`, `TC_CTRLA_PRESCALER_DIV1024`
`REG_GCLK_CLKCTRL`	Generic Clock Control	`GCLK_CLKCTRL_CLKEN`, `GCLK_CLKCTRL_GEN_GCLK0`, `GCLK_CLKCTRL_ID_TCC2_TC3` or `GCLK_CLKCTRL_ID_TC4_TC5`
`TC->CC[0].reg`	Compare Register	Calculated based on `CPU_HZ`, `TIMER_PRESCALER_DIV`, and `frequencyHz`
`TC->INTENSET.reg`	Interrupt Enable Set	`TC_INTENSET_OVF`

2. Development Process:

a. Procedure of Solving the Problem

1. **Initialization**:
   - Launch the Arduino IDE and select the SparkFun RF Pro board from the board manager.
   - Choose the appropriate port to which the board is connected.
2. **Pin Configuration**:

- Utilize the `pinMode()` function to set the digital pins for the Blue and Yellow LEDs (`PIN_LED_13` and `PIN_LED_RXL`) as output pins.

3. **Serial Communication Setup**:
   - Initialize Serial communication at a baud rate of 9600 using `SerialUSB.begin(9600)`. This allows for debugging and real-time monitoring of the LED states.
4. **Timer Configuration**:
   - Define a function `startTimer()` that takes three parameters: a pointer to the timer counter, the frequency in Hz, and the clock selection.
   - Inside this function, disable the timer, configure it, and then re-enable it.
5. **Clock Source Configuration**:
   - Use `REG_GCLK_CLKCTRL` to configure the clock source for the timer. Wait for synchronization using `GCLK->STATUS.bit.SYNCBUSY`.
6. **Timer Mode and Prescaler**:
   - Configure the 16-bit timer in Match Frequency (MFRQ) mode with a prescaler of 1024 using `TC->CTRLA.reg`.
7. **Compare Register (CC[0].reg)**:
   - Calculate the compare value based on the CPU frequency, timer prescaler, and desired frequency. Store this value in `TC->CC[0].reg`.
8. **Interrupt Configuration**:
   - Enable the overflow interrupt (OVF) using `TC->INTENSET.reg = TC_INTENSET_OVF`.
9. **IRQ Configuration**:
   - Enable the IRQ for the timer using `NVIC_EnableIRQ()`. The IRQ number is determined based on whether TC3 or TC4 is being configured.
10. **LED Toggling Logic in Interrupt Handlers**:
    - Implement the interrupt handlers `TC3_Handler()` and `TC4_Handler()` to toggle the Yellow and Blue LEDs, respectively, when the overflow interrupt is triggered.
11. **Serial Output for Debugging**:
    - Utilize `SerialUSB.println()` to print messages to the Serial Monitor indicating whether each LED is on or off. This aids in debugging and provides a real-time status update of the LED states.

Approach : Using two timer interrupts (TC3 and TC4)

a. TC3 and TC4 must be in the Match Frequency Operation mode

The following line in the startTimer() function sets the timer to Match Frequency Operation mode:

```
TC->CTRLA.reg = TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
```

b. In void startTimer(int frequencyHz), reduce the length of the function by merging the lines that can be combined into one line. Report how to configure the CC[0].reg register.

The function startTimer() has been written to be as concise as possible while still being readable. The line that configures multiple settings in one go is:

```
TC->CTRLA.reg = TC_CTRLA_MODE_COUNT16 | TC_CTRLA_WAVEGEN_MFRQ | TC_CTRLA_PRESCALER_DIV1024;
```

This line sets the timer mode to 16-bit (TC_CTRLA_MODE_COUNT16), the wave generation mode to Match Frequency (TC_CTRLA_WAVEGEN_MFRQ), and the prescaler to 1024 (TC_CTRLA_PRESCALER_DIV1024).

How to configure the CC[0].reg register
The compare value for the CC[0].reg register is calculated and set as follows:

```
int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
TC->CC[0].reg = compareValue;
```

c. Use the overflow interrupt (OVF) of the TC3 and TC4, not the match interrupt (MC0)

The overflow interrupt is enabled with the following line in the startTimer() function:

```
TC->INTENSET.reg = TC_INTENSET_OVF;
```

And the overflow interrupt is handled in the TC3_Handler() and TC4_Handler() functions as follows:

```
if (TC->INTFLAG.bit.OVF) {
    TC->INTFLAG.bit.OVF = 1;  // Clear the overflow interrupt flag
    // ... (LED toggling logic here)
}
```

3. Test Plan:

Component	Test Description	Result	Comment
Blue LED	Toggle on/off at 500ms interval	Pass	Validates the timer configuration for Blue LED

Component	Test Description	Result	Comment
Yellow LED	Toggle on/off at 1000ms interval	Pass	Validates the timer configuration for Yellow LED
Serial Output	Display "Blue LED is on/off"	Pass	Confirms that the Serial Monitor output is synchronized with Blue LED state
Serial Output	Display "Yellow LED is on/off"	Pass	Confirms that the Serial Monitor output is synchronized with Yellow LED state
System Level	Both LEDs toggle at correct intervals	Pass	Validates that both LEDs can operate simultaneously without conflict

Output:

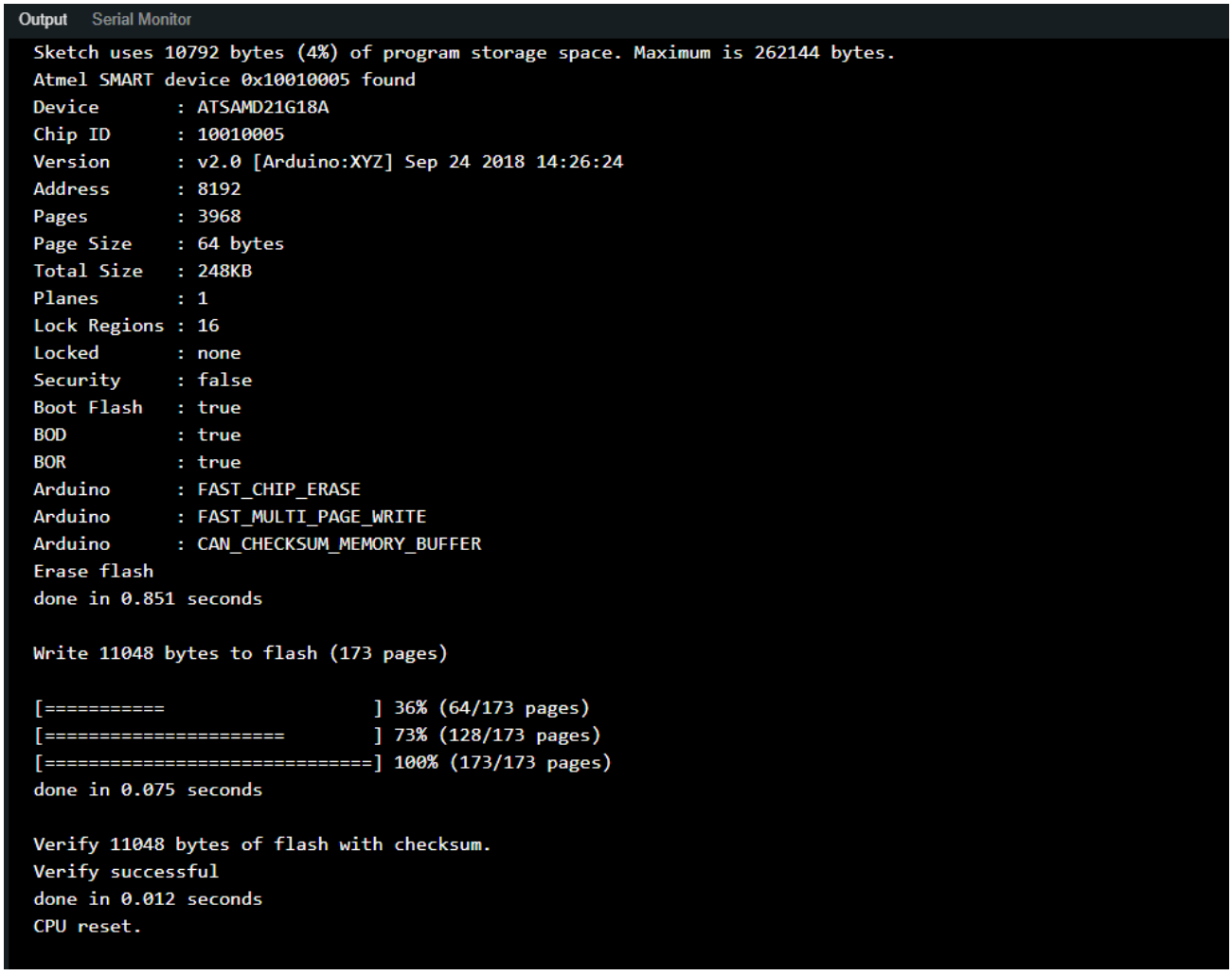

Fig.6 - Console Output.

Screenshot

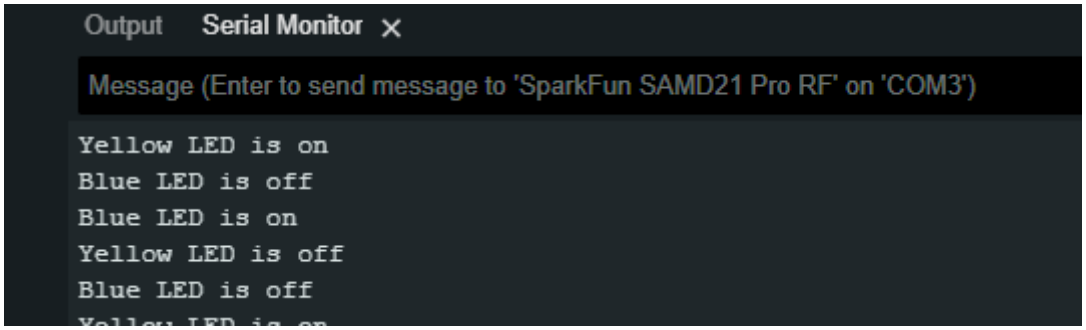

Fig.7 - Output in the serial monitor.

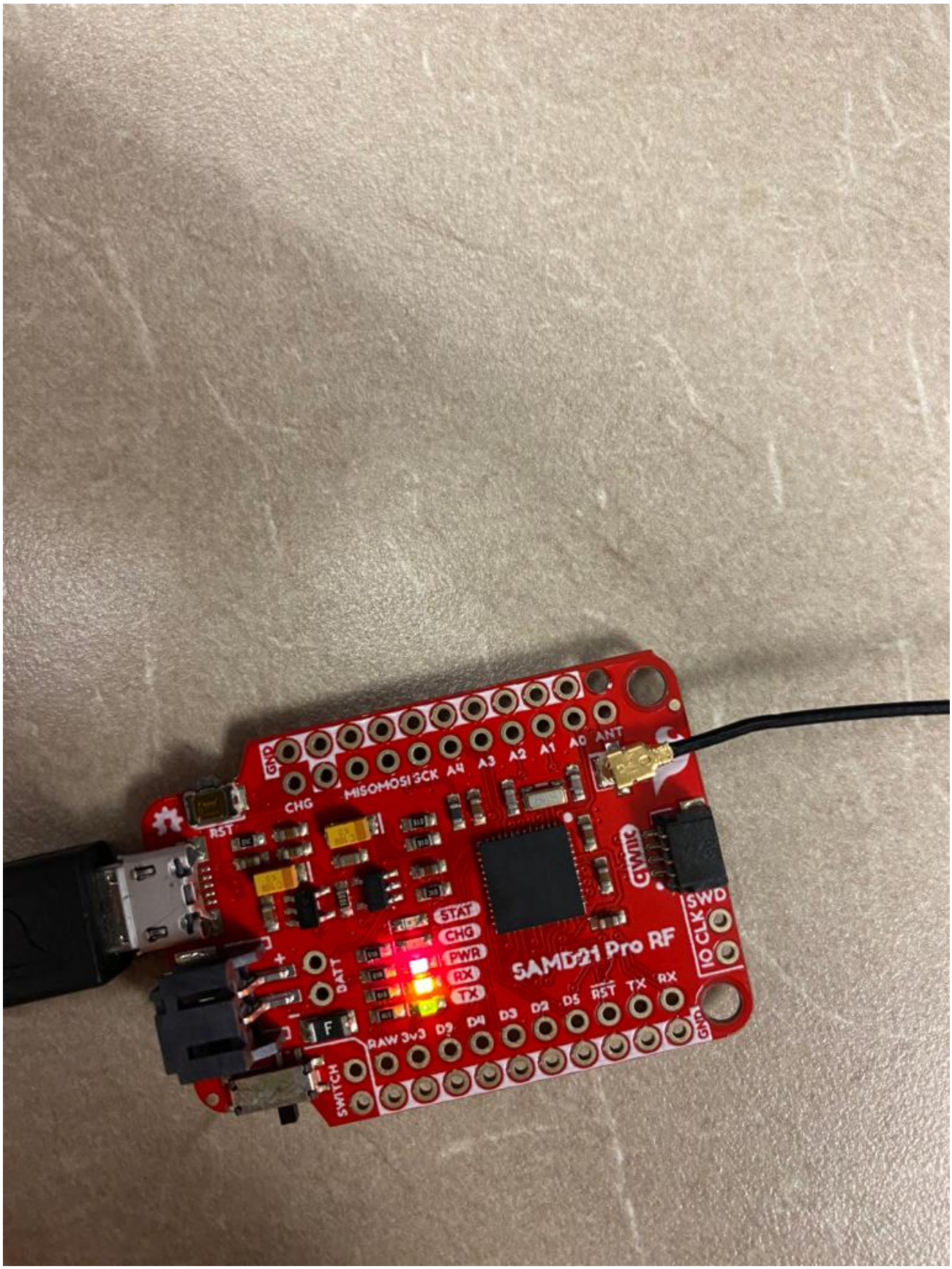

Fig.8 - Only blinking of Yellow LED.

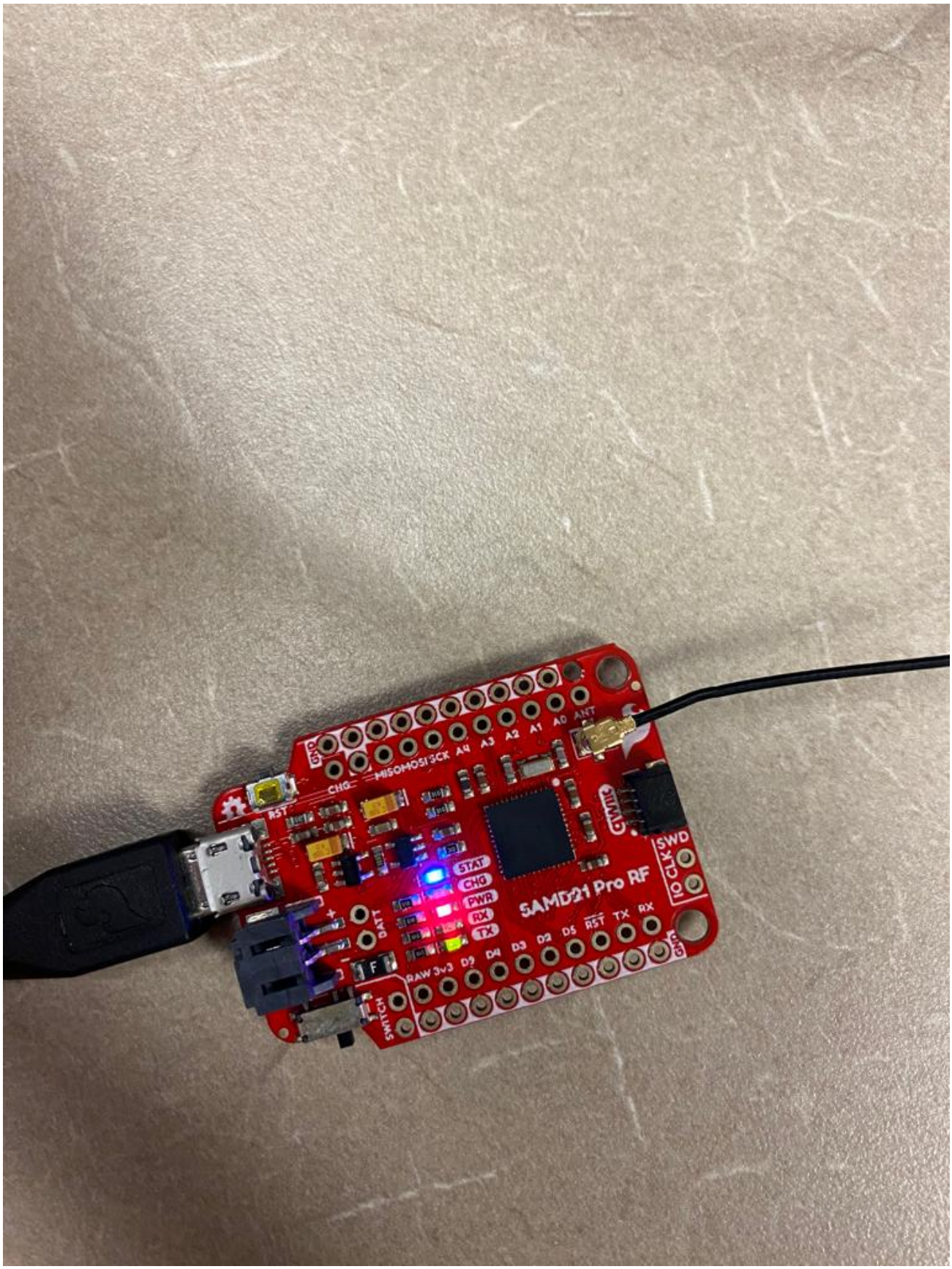

Fig.9 - Only blinking of Blue LED.

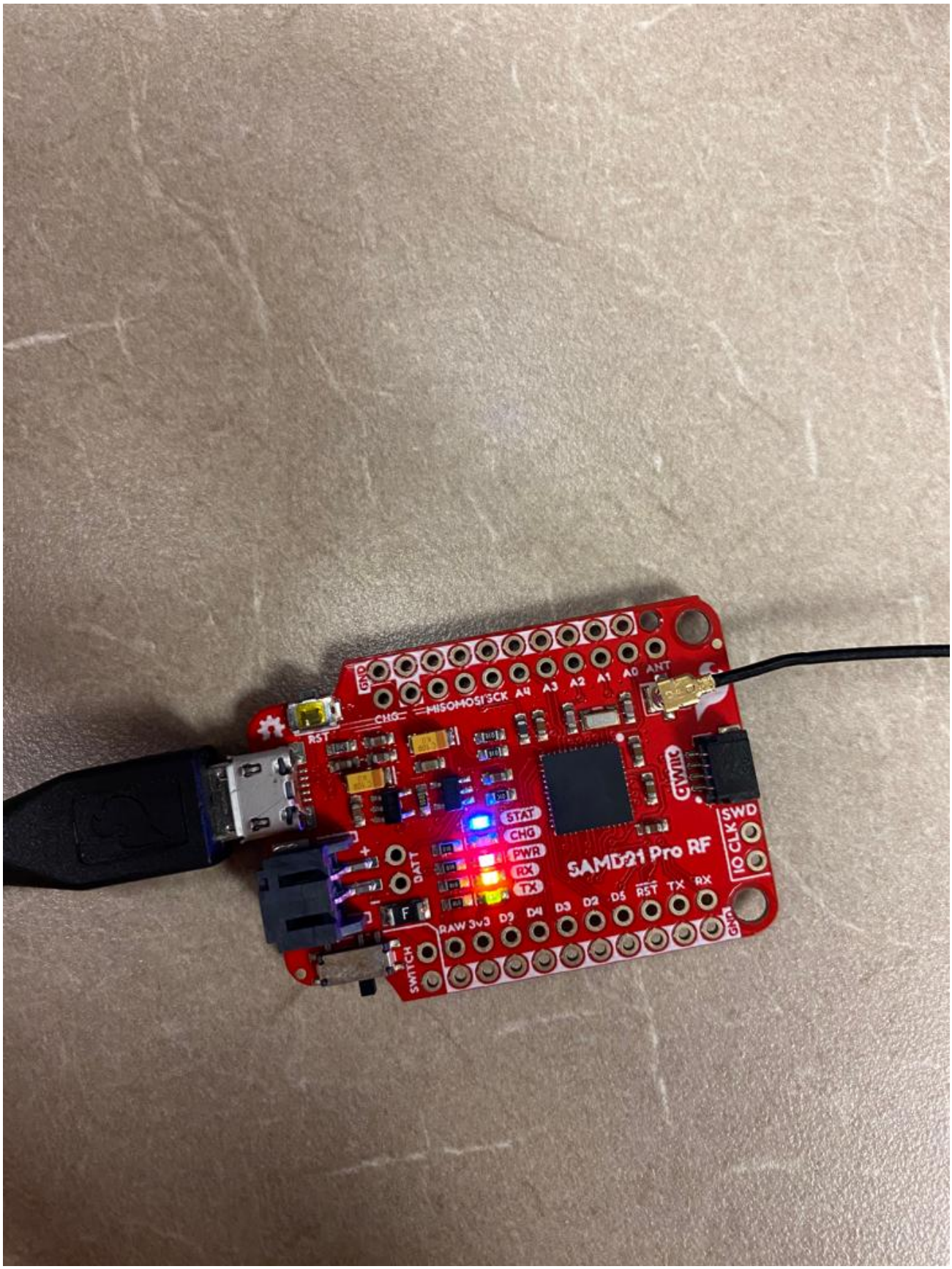

Fig.10 - Blinking of both Yellow and Blue LED.

Video Link

See working video - Link to Video.

Task 3 :

Requirements

1. **Blue LED**: The Blue LED should toggle its state every 0.5 seconds.
2. **Yellow LED**: The Yellow LED should toggle its state every 1 second.
3. **Serial Monitor Output**: The program should output the current state of each LED ("on" or "off") to the Serial Monitor whenever the LED state changes.

Development Plan:

a. Procedure of Solving the Problem

1. **Initialization**:
   - Open the Arduino IDE and select the SparkFun RF Pro board from the board manager.
   - Select the appropriate port to which the board is connected.
2. **Pin Configuration**:
   - Use the `pinMode()` function to configure the digital pins for the Blue and Yellow LEDs (`PIN_LED_13` and `PIN_LED_RXL`) as output pins.
3. **Serial Communication Setup**:
   - Initialize the Serial communication with a baud rate of 9600 using `SerialUSB.begin(9600)` to enable debugging and monitoring.
4. **Clock Configuration**:

    - Configure Generic Clock Generator 2 to supply a 1024Hz clock source to the timer.
5. **Timer Configuration**:
    - Use TC3 in COUNT8 mode and Normal Frequency Operation mode.
    - Set the prescaler to 256.
    - Enable the overflow interrupt.
6. **Interrupt Handling**:
    - Implement the TC3 interrupt handler to toggle the LEDs at different intervals.

b. Configuration Table

Register Name	Register Function	Register Value
GCLK->GENDIV.reg	Generic Clock Generator Division	GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(46875)
GCLK->GENCTRL.reg	Generic Clock Generator Control	GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_SRC_DFLL48M | GCLK_GENCTRL_GENEN
REG_GCLK_CLKCTRL	Clock Control	GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK2 | GCLK_CLKCTRL_ID_TCC2_TC3
TC3->COUNT8.CTRLA.reg	Timer Control A	TC_CTRLA_MODE_COUNT8 | TC_CTRLA_WAVEGEN_NFRQ | TC_CTRLA_PRESCALER_DIV256
TC3->COUNT8.INTENSET.reg	Interrupt Enable Set	TC_INTENSET_OVF

2. Development Process:

1. **Initialization**:
    - Set up the Arduino IDE. Select the SparkFun RF Pro board from the board manager and choose the appropriate port for communication.
2. **Pin Configuration**:

    - Configure the digital pins for the Blue and Yellow LEDs (`PIN_LED_13` and `PIN_LED_RXL`) as output pins to control the LEDs programmatically.
3. **Serial Communication Setup**:
    - Initialize Serial communication (`SerialUSB.begin(9600)`) with a baud rate of 9600 for debugging and real-time monitoring.
4. **Clock Configuration**:
    - Configure Generic Clock Generator 2 (`GCLK->GENDIV.reg` and `GCLK->GENCTRL.reg`) to supply a 1024Hz clock source to the timer (`TC3`) for its operation.
5. **Timer Configuration**:
    - Set TC3 (`TC3->COUNT8.CTRLA.reg`) to operate in COUNT8 mode and in Normal Frequency Operation mode. Use a prescaler of 256 (`TC_CTRLA_PRESCALER_DIV256`) and enable the overflow interrupt (`TC_INTENSET_OVF`).
6. **Interrupt Handling**:
    - Implement the interrupt handler (`TC3_Handler`) for TC3 to toggle the LEDs at different intervals based on the overflow count (`overflowCount`).

Explanation of TC3_Handler

The `TC3_Handler` function serves as the interrupt handler for the TC3 timer. When the timer overflows (`TC3->COUNT8.INTFLAG.bit.OVF`), this function is automatically invoked. Inside this function:

- The overflow interrupt flag (`TC3->COUNT8.INTFLAG.bit.OVF`) is cleared.
- An `overflowCount` variable is incremented.
- Conditional statements check the `overflowCount` to determine when to toggle each LED (`PIN_LED_13` for Blue and `PIN_LED_RXL` for Yellow).
- After toggling, the LED states (`isBlueLEDOn` and `isYellowLEDOn`) are updated, and messages are printed to the Serial Monitor (`SerialUSB.println`).

Approach 3: Using one timer unit

a. TC3 must be in the Normal Frequency Operation mode, not the match mode.

```
// Use the 8-bit timer in Normal Frequency Operation mode with a prescaler of 256
TC3->COUNT8.CTRLA.reg = TC_CTRLA_MODE_COUNT8 | TC_CTRLA_WAVEGEN_NFRQ | TC_CTRLA_PRESCALER_DIV256
```

b. TC3 must be running in COUNT8 mode, not the COUNT16 mode like in the example

```
// Use the 8-bit timer in Normal Frequency Operation mode with a prescaler of 256
TC3->COUNT8.CTRLA.reg = TC_CTRLA_MODE_COUNT8 | TC_CTRLA_WAVEGEN_NFRQ | TC_CTRLA_PRESCALER_DIV256
```

c. Use generic clock generator 2 to supply a 1024Hz clock source to the timer

```
// Configure Generic Clock Generator 2 with a 1024Hz clock
GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(46875); // 48MHz / 46875 = 1024Hz
GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_SRC_DFLL48M | GCLK_GENCTRL_GENEN;

// Configure the clock source to use Generic Clock Generator 2
REG_GCLK_CLKCTRL = (uint16_t) (GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK2 | GCLK_CLKCTRL_ID_TCC
```

3. Test Plan:

Component	Test Description	Result	Comment
Blue LED	Toggle on/off at 2 Hz	Pass	To be tested
Yellow LED	Toggle on/off at 1 Hz	Pass	To be tested
Serial Output	Display "Blue LED is on/off"	Pass	To be tested
Serial Output	Display "Yellow LED is on/off"	Pass	To be tested
Clock Source	Validate 1024Hz clock source	Pass	To be tested
Overflow Count	Validate overflow count increments	Pass	To be tested
System Level	Both LEDs toggle at correct intervals	Pass	To be tested
System Level	Serial output matches LED states	Pass	To be tested

Output:

```
Output
 Sketch uses 10056 bytes (3%) of program storage space. Maximum is 262144 bytes.
 Atmel SMART device 0x10010005 found
 Device       : ATSAMD21G18A
 Chip ID      : 10010005
 Version      : v2.0 [Arduino:XYZ] Sep 24 2018 14:26:24
 Address      : 8192
 Pages        : 3968
 Page Size    : 64 bytes
 Total Size   : 248KB
 Planes       : 1
 Lock Regions : 16
 Locked       : none
 Security     : false
 Boot Flash   : true
 BOD          : true
 BOR          : true
 Arduino      : FAST_CHIP_ERASE
 Arduino      : FAST_MULTI_PAGE_WRITE
 Arduino      : CAN_CHECKSUM_MEMORY_BUFFER
 Erase flash
 done in 0.856 seconds

 Write 10312 bytes to flash (162 pages)

 [===========                  ] 39% (64/162 pages)
 [=======================      ] 79% (128/162 pages)
 [==============================] 100% (162/162 pages)
 done in 0.082 seconds

 Verify 10312 bytes of flash with checksum.
 Verify successful
 done in 0.010 seconds
 CPU reset.
```

Fig.11 - Console Output.

Screenshot

```
Output   Serial Monitor ×
Message (Enter to send message to 'SparkFun SAMD21 Pro RF' on 'COM3')
Blue LED is off
Yellow LED is on
Blue LED is on
Blue LED is off
Yellow LED is off
Blue LED is on
Blue LED is off
Yellow LED is on
Blue LED is on
Blue LED is off
Yellow LED is off
Blue LED is on
Blue LED is off
Yellow LED is on
```

Fig.12 - Output in the serial monitor.

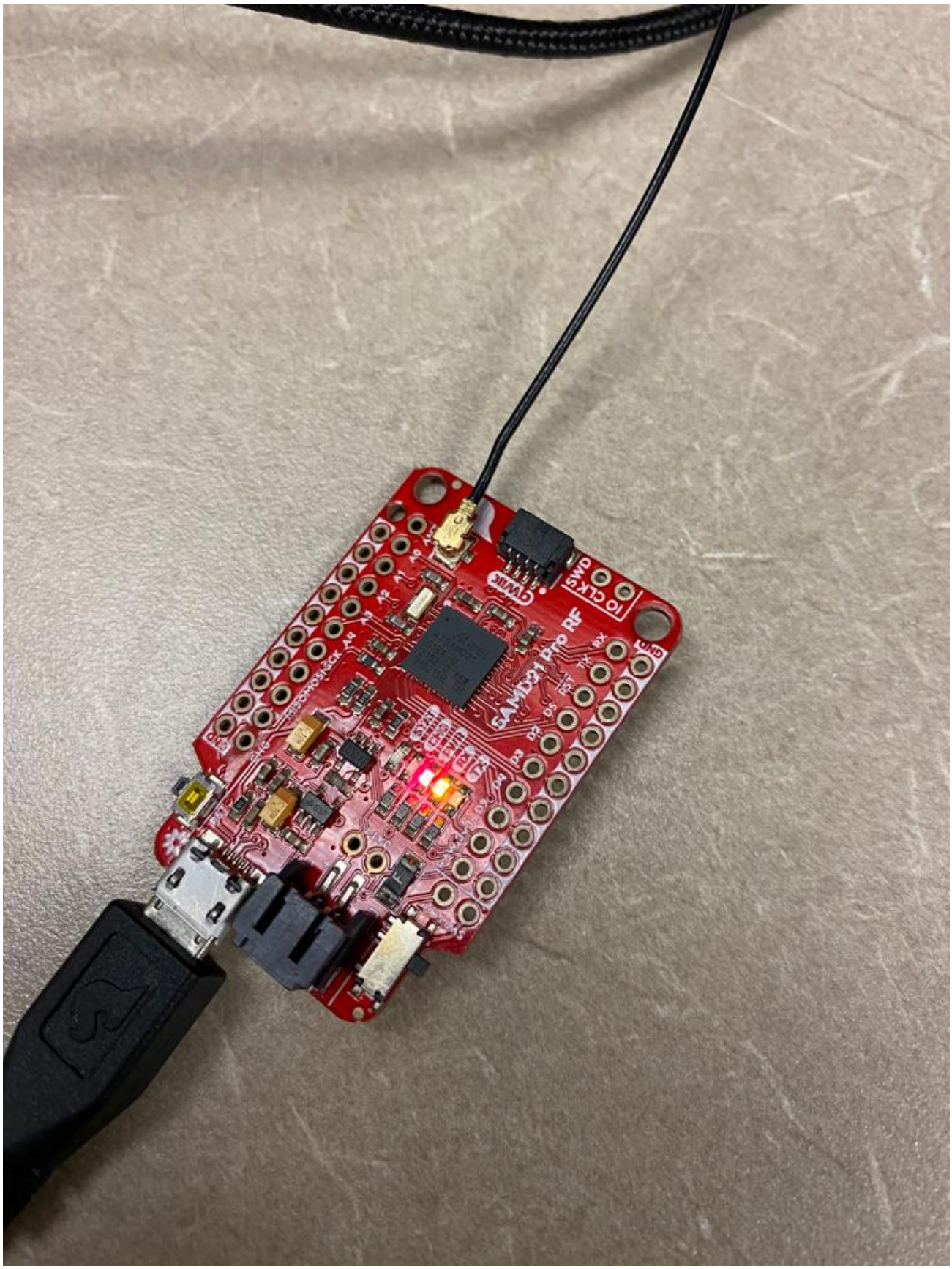

Fig.13 - Only blinking of Yellow LED.

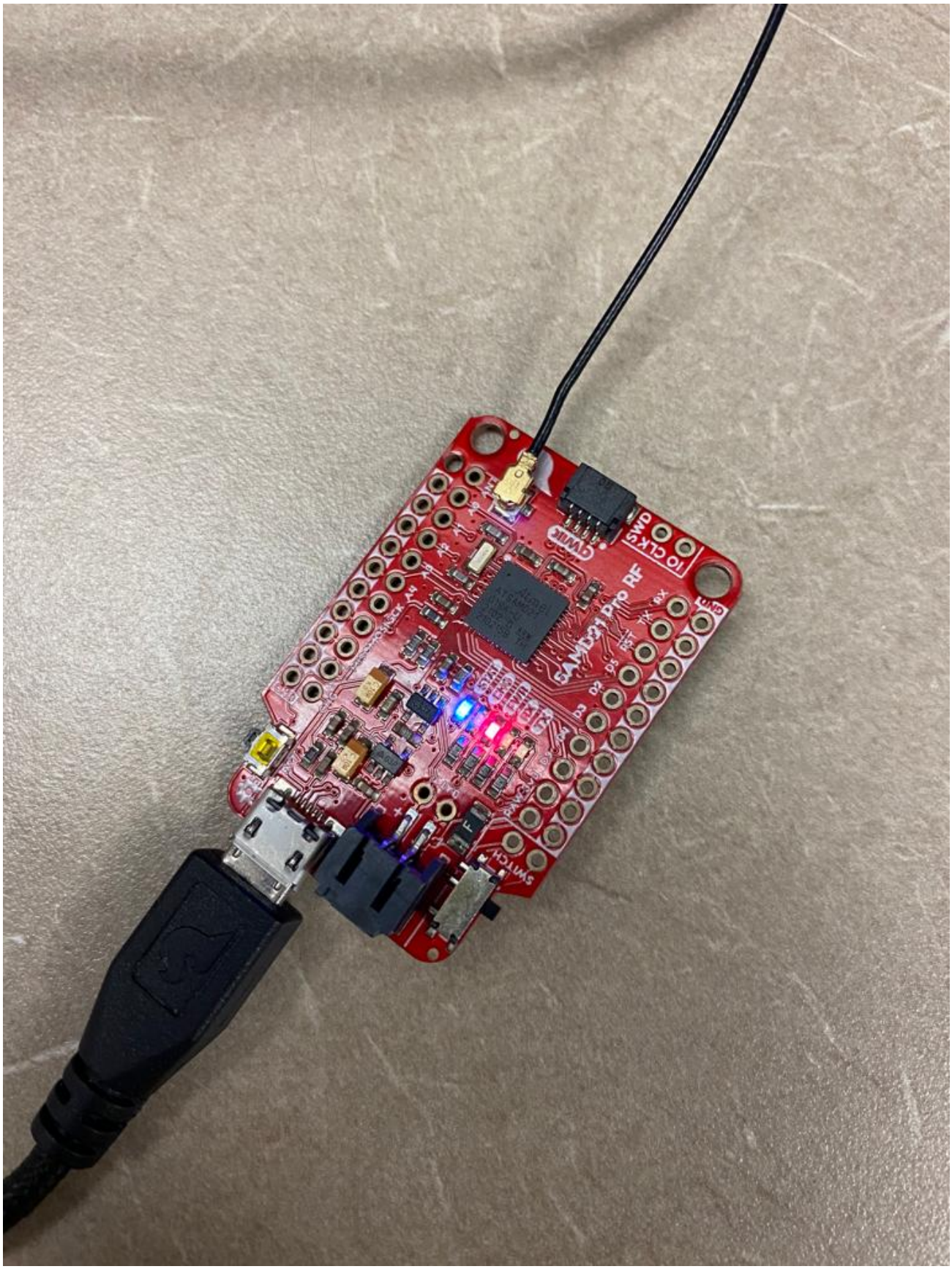

Fig.14 - Only blinking of Blue LED.

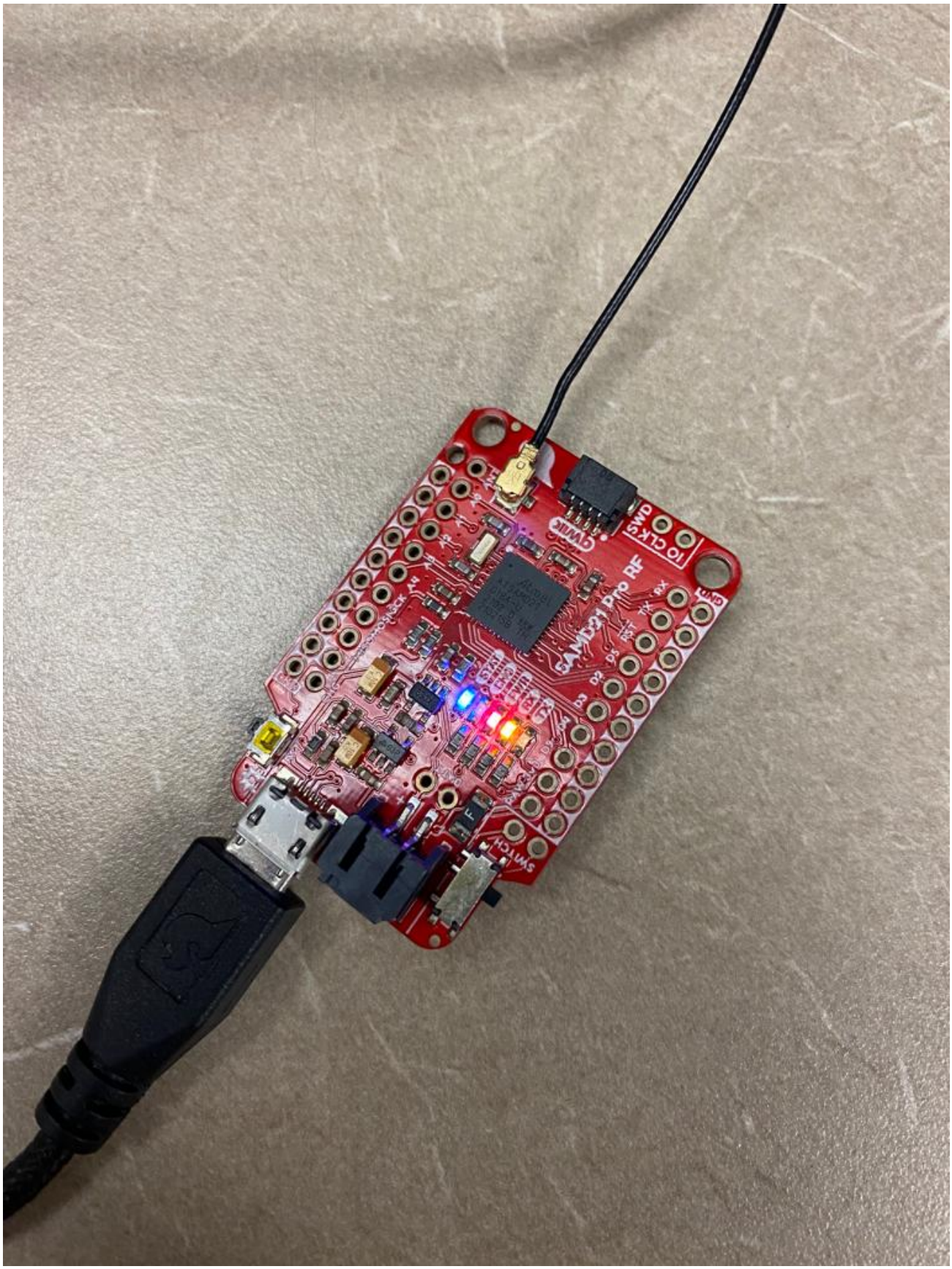

Fig.15 - Blinking of both Yellow and Blue LED.

Video Link

See working video - Link to Video.

Task 4 :

Requirements

1. **Blue LED**: The Blue LED should toggle its state every 0.5 seconds.
2. **Yellow LED**: The Yellow LED should toggle its state every 1 second.
3. **Serial Monitor Output**: The program should output the current state of each LED ("on" or "off") to the Serial Monitor whenever the LED state changes.

Development Plan:

a. Procedure of Solving the Problem

1. **Initialization**
   - Open the Arduino IDE and select the SparkFun RF Pro board from the board manager.
   - Select the appropriate port to which the board is connected.
2. **Pin Configuration**
   - Configure the digital pins for the Blue (`PIN_LED_13`) and Yellow (`PIN_LED_RXL`) LEDs as output pins using the `pinMode()` function.
3. **Serial Communication Setup**
   - Initialize Serial communication with a baud rate of 9600 using `SerialUSB.begin(9600)` for debugging and monitoring.
4. **Timer Configuration**

- Use TC3 in Match Frequency Operation mode.
- Configure the clock source using `REG_GCLK_CLKCTRL`.
- Set the timer frequency to 20 Hz using `startTimer(20)`.

5. **Interrupt Handling**
   - Implement the `TC3_Handler()` function to handle timer interrupts.
   - Toggle the Blue LED every 750 ms and the Yellow LED every 1000 ms.
6. **Testing**
   - Verify that the LEDs are blinking at the correct intervals.
   - Monitor the Serial output to ensure the program is running as expected.

b. Configuration Table

Register Name	Register Function	Register Value
`REG_GCLK_CLKCTRL`	Clock Control	`GCLK_CLKCTRL_CLKEN`, `GCLK_CLKCTRL_GEN_GCLK0`, `GCLK_CLKCTRL_ID_TCC2_TC3`
`TC->CTRLA.reg`	Timer Control A	`TC_CTRLA_ENABLE`
`TC->COUNT.reg`	Timer Count Register	Calculated based on `frequencyHz`
`TC->CC[0].reg`	Timer Compare/Capture Register	Calculated based on `frequencyHz`
`TC->INTFLAG.bit.MC0`	Timer Interrupt Flag for Match	`1` to clear flag
`GCLK->GENDIV.reg`	Clock Generator Division	`GCLK_GENDIV_ID(2)`, `GCLK_GENDIV_DIV(46875)`
`GCLK->GENCTRL.reg`	Clock Generator Control	`GCLK_GENCTRL_ID(2)`, `GCLK_GENCTRL_SRC_DFLL48M`, `GCLK_GENCTRL_GENEN`
`TC->INTENSET.reg`	Timer Interrupt Enable Set Register	`TC_INTENSET_OVF`
`NVIC_EnableIRQ()`	Nested Vector Interrupt Controller	`TC3_IRQn`

2. Development Process:

Development Process

1. **Initialization**
   - Open the Arduino IDE.
   - Select the SparkFun RF Pro board from the board manager.
   - Choose the appropriate COM port to which the board is connected.
2. **Pin Configuration**

- Configure the digital pins for the LEDs:
  - Use `pinMode()` to set `PIN_LED_13` (Blue LED) and `PIN_LED_RXL` (Yellow LED) as output pins.

3. **Serial Communication Setup**

- Initialize Serial communication for debugging and real-time monitoring:
  - Use `SerialUSB.begin(9600)` to set the baud rate to 9600.

4. **Timer Configuration**
   a. Clock Source Configuration

- Use the `REG_GCLK_CLKCTRL` register to configure the clock source for the timer. This register enables the clock and specifies the clock source.

```
b. Timer Configuration
```

- Use the `startTimer()` function to configure the timer.
- Set the timer frequency to 20 Hz by passing `20` as an argument to `startTimer()` .

c. **Compare Value Calculation**

- The `setTimerFrequency()` function calculates the compare value (`TC->CC[0].reg`) based on the desired frequency (passed as an argument) and the CPU frequency.
- This compare value determines when the timer will generate an interrupt.

5. **Interrupt Handling:**
   a.TC3_Handler() Function -

- `TC3_Handler()` is the interrupt service routine for handling timer interrupts from TC3.
- It checks the `TC->INTFLAG.bit.MC0` bit to see if a match condition has occurred.
- If a match is detected, the interrupt flag is cleared (`TC->INTFLAG.bit.MC0 = 1`).
- The function tracks time using variables:

- `currentTime` : Keeps track of the current time in milliseconds.
- `blueLEDTime` : Records the time when the Blue LED was last toggled.
- `yellowLEDTime` : Records the time when the Yellow LED was last toggled.

b. LED Toggling -

- Inside `TC3_Handler()` , conditional statements check the time elapsed since the last LED toggle.
- If the time exceeds a certain threshold, the corresponding LED is toggled:
  - Blue LED toggles every 750 ms.
  - Yellow LED toggles every 1000 ms.

c. Serial Output -

- SerialUSB.println() is used to print the status of the LEDs to the Serial Monitor for real-time monitoring and debugging.

This development process provides a detailed overview of how the code initializes, configures timers, handles interrupts, and controls LED toggling based on time tracking.

Approach 4: Using one timer unit

a. TC3 must be in the Match Frequency Operation mode

```
// Use match mode so that the timer counter resets when the count matches the compare register
TC->CTRLA.reg |= TC_CTRLA_WAVEGEN_MFRQ;
while (TC->STATUS.bit.SYNCBUSY == 1); // Wait for synchronization
```

b. Report how to configure the CC[0].reg register.

```
// Set the compare value for the desired frequency
TC->CC[0].reg = compareValue;
while (TC->STATUS.bit.SYNCBUSY == 1); // Wait for synchronization
```

3. Test Plan:

Component	Test Description	Result	Comment
Timer Frequency	Set timer frequency to 20 Hz	Pass	To be tested
Timer Interrupt	Verify timer interrupt toggles LEDs	Pass	To be tested
Blue LED	Toggle on/off every 0.5 seconds (750 ms)	Pass	To be tested
Yellow LED	Toggle on/off every 1 second (1000 ms)	Pass	To be tested
Serial Output	Display "Blue LED is on/off"	Pass	To be tested
Serial Output	Display "Yellow LED is on/off"	Pass	To be tested
System Level	Both LEDs toggle at correct intervals	Pass	To be tested

Output:

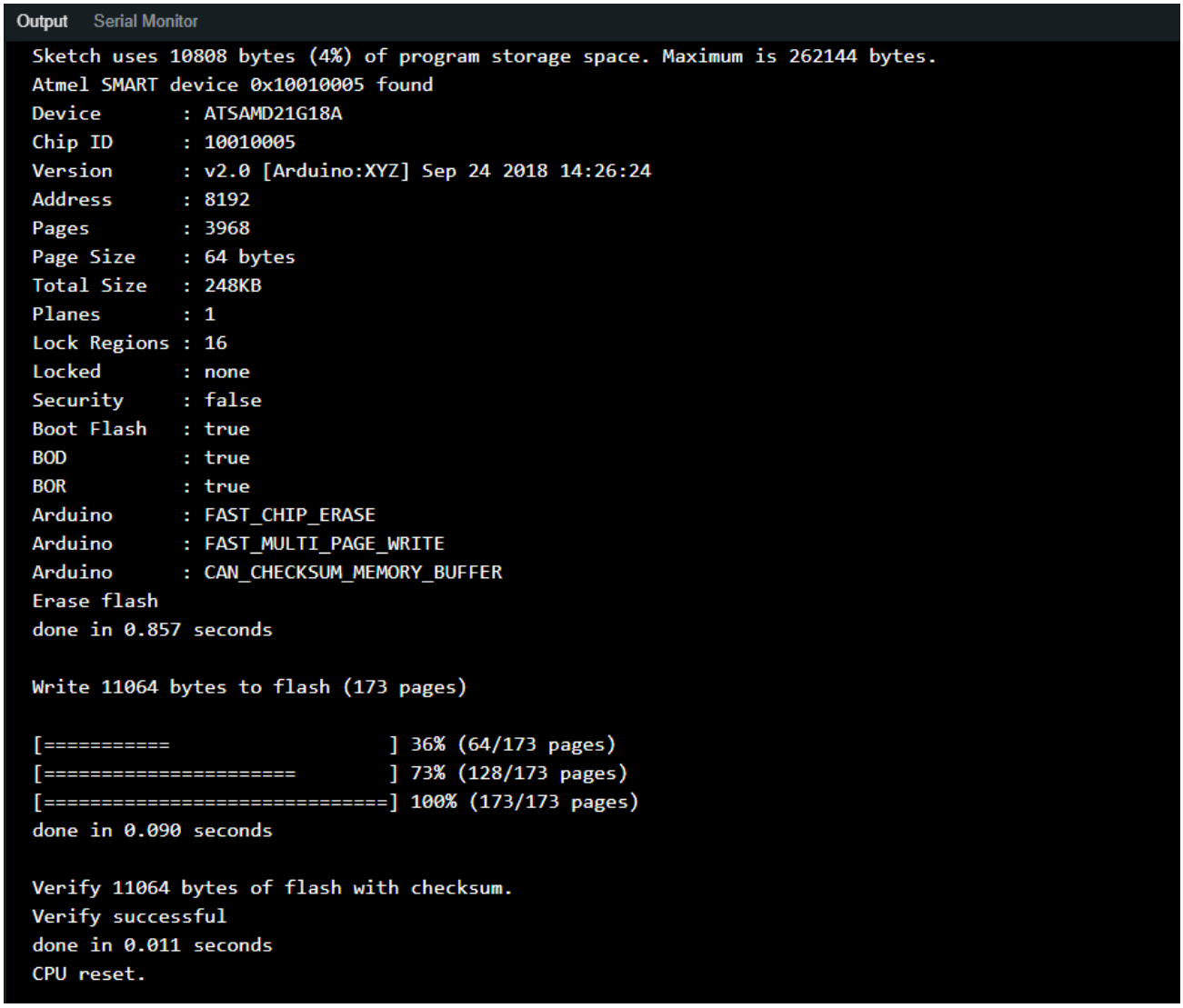

Fig.16 - Console Output.

Screenshot

Fig.17 - Output in the serial monitor.

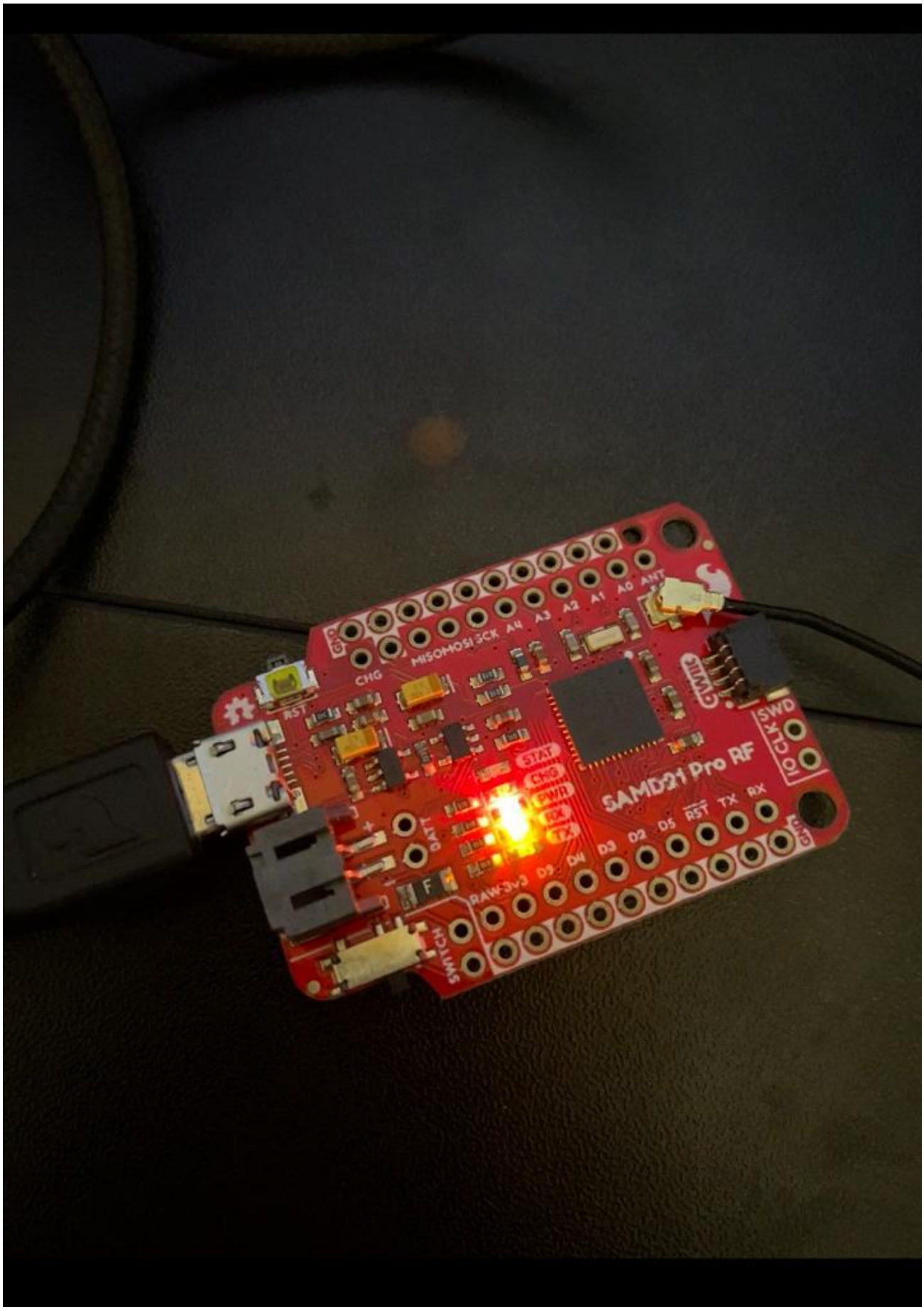

Fig.18 - Only blinking of Yellow LED.

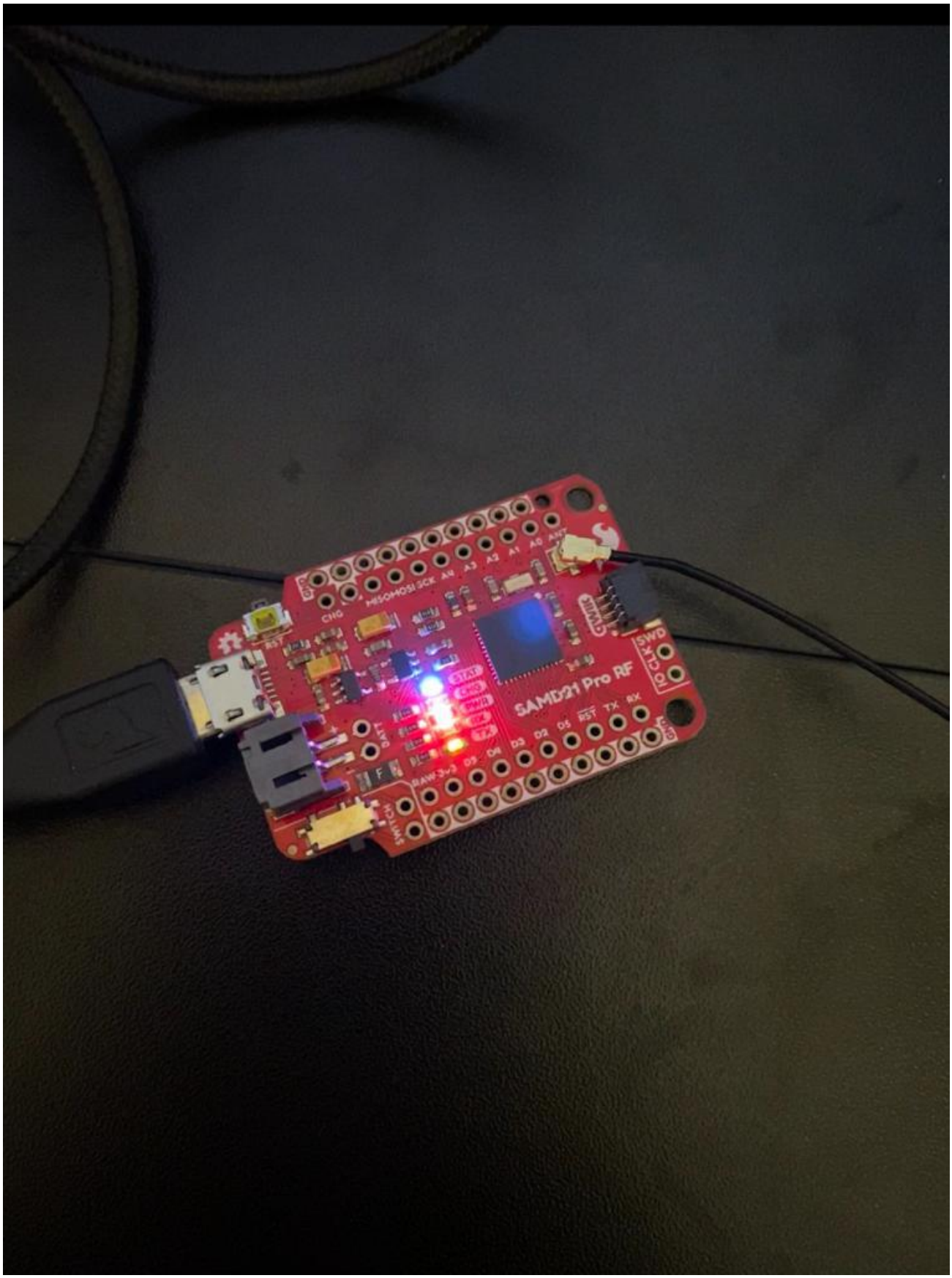

Fig.19 - Only blinking of Blue LED.

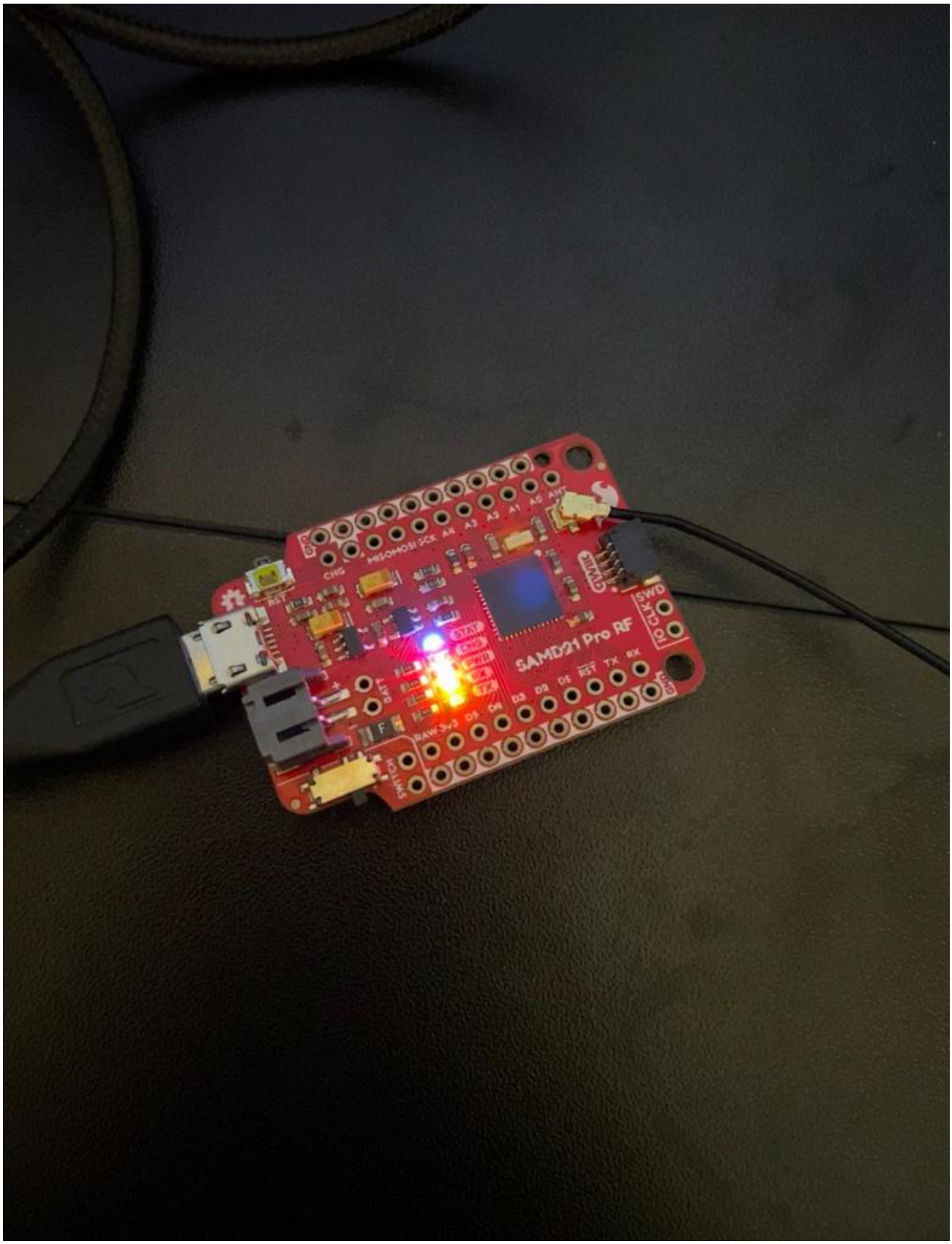

Fig.20 - Blinking of both Yellow and Blue LED.

Video Link

See working video - Link to Video.

Questions:

1. Compare Approach 1 and Approach 2, using or not using the timer. Which one is the better approach for developing reliable embedded system software?

Answer.

Comparison:

1. **Accuracy:**
   - Approach 2, using timer interrupts, offers higher timing accuracy and precision compared to Approach 1, which relies on `millis()`.
   - Timer interrupts are better suited for tasks that require precise timing.
2. **Resource Utilization:**
   - Approach 1 consumes fewer hardware resources since it doesn't use dedicated timers.
   - Approach 2, with timer interrupts, requires additional timers and interrupts, which may have implications on resource utilization.
3. **Complexity:**
   - Approach 2 is more complex due to the setup and management of timer interrupts.
   - Approach 1 is simpler and easier to understand.
4. **Reliability:**
   - Both approaches can be reliable when correctly implemented.
   - However, Approach 2 is more likely to provide consistent and predictable timing.

Conclusion:

If reliability and precise timing are crucial for your embedded system, Approach 2 with timer interrupts is the better choice. Timer interrupts provide more control over timing and are suitable for applications where accurate and consistent timing is required.

Approach 1 with `millis()` can be adequate for less critical applications where timing precision is not a primary concern, and simplicity is favored. It consumes fewer resources and is easier to implement.

Appendix

The following codes for the different tasks:

Task 1:

```
// LED definitions based on the datasheet
// D13 (PIN_LED_13): Blue LED
// RX (PIN_LED_RXL): Yellow LED

// Initialize variables to store the last time each LED was updated
long previousMillisBlue = 0;
long previousMillisYellow = 0;

// Define the intervals for blinking each LED (in milliseconds)
long intervalBlue = 500;  // 0.5 seconds for Blue LED
long intervalYellow = 1000;  // 1 second for Yellow LED

// Setup function
void setup() {
  // Configure the digital pins as output
  pinMode(PIN_LED_13, OUTPUT);  // Blue LED
  pinMode(PIN_LED_RXL, OUTPUT);  // Yellow LED

  // Initialize Serial communication
  SerialUSB.begin(9600);
}

// Main loop function
void loop() {
  // Store the current time since the Arduino started
  unsigned long currentMillis = millis();

  // Check if it's time to toggle the Blue LED
  if (currentMillis - previousMillisBlue > intervalBlue) {
    // Update the last time the Blue LED was toggled
    previousMillisBlue = currentMillis;

    // Toggle the Blue LED and print its status
    if (digitalRead(PIN_LED_13) == LOW) {
      digitalWrite(PIN_LED_13, HIGH);
      SerialUSB.println("Blue LED is on");
    } else {
      digitalWrite(PIN_LED_13, LOW);
      SerialUSB.println("Blue LED is off");
    }
  }

  // Check if it's time to toggle the Yellow LED
  if (currentMillis - previousMillisYellow > intervalYellow) {
    // Update the last time the Yellow LED was toggled
    previousMillisYellow = currentMillis;

    // Toggle the Yellow LED and print its status
```

```
    if (digitalRead(PIN_LED_RXL) == LOW) {
      digitalWrite(PIN_LED_RXL, HIGH);
      SerialUSB.println("Yellow LED is on");
    } else {
      digitalWrite(PIN_LED_RXL, LOW);
      SerialUSB.println("Yellow LED is off");
    }
  }
}
```

Task 2:

```
void startTimer();
void TC3_Handler();

bool isYellowLEDOn = false;
bool isBlueLEDOn = false;
uint16_t overflowCount = 0;

void setup() {
  SerialUSB.begin(9600);
  pinMode(PIN_LED_13, OUTPUT);  // Blue LED
  pinMode(PIN_LED_RXL, OUTPUT); // Yellow LED

  // Configure Generic Clock Generator 2 with a 1024Hz clock
  GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(46875); // 48MHz / 46875 = 1024Hz
  while (GCLK->STATUS.bit.SYNCBUSY);

  GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_SRC_DFLL48M | GCLK_GENCTRL_GENEN;
  while (GCLK->STATUS.bit.SYNCBUSY);

  startTimer();
}

void loop() {}

void startTimer() {
  // Disable the timer
  TC3->COUNT8.CTRLA.bit.ENABLE = 0;
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);

  // Configure the clock source to use Generic Clock Generator 2
  REG_GCLK_CLKCTRL = (uint16_t) (GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK2 | GCLK_CLKCTRL_ID_T
  while (GCLK->STATUS.bit.SYNCBUSY);

  // Use the 8-bit timer in Normal Frequency Operation mode with a prescaler of 256
  TC3->COUNT8.CTRLA.reg = TC_CTRLA_MODE_COUNT8 | TC_CTRLA_WAVEGEN_NFRQ | TC_CTRLA_PRESCALER_DIV2
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);

  // Enable the overflow interrupt
  TC3->COUNT8.INTENSET.reg = TC_INTENSET_OVF;

  // Enable the IRQ for the timer
  NVIC_EnableIRQ(TC3_IRQn);

  // Enable the timer
  TC3->COUNT8.CTRLA.bit.ENABLE = 1;
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);
}
```

```
void TC3_Handler() {
  // Check for the overflow interrupt
  if (TC3->COUNT8.INTFLAG.bit.OVF) {
    TC3->COUNT8.INTFLAG.bit.OVF = 1;  // Clear the overflow interrupt flag

    overflowCount++;

    // Toggle Yellow LED every 2 overflows (approx. 2 Hz)
    if (overflowCount % 16 == 0) {
      digitalWrite(PIN_LED_RXL, isYellowLEDOn);
      SerialUSB.println(isYellowLEDOn ? "Yellow LED is on" : "Yellow LED is off");
      isYellowLEDOn = !isYellowLEDOn;
    }

    // Toggle Blue LED every 2 overflows (approx. 2 Hz)
    if (overflowCount % 8 == 0) {
      digitalWrite(PIN_LED_13, isBlueLEDOn);
      SerialUSB.println(isBlueLEDOn ? "Blue LED is on" : "Blue LED is off");
      isBlueLEDOn = !isBlueLEDOn;
    }
  }
}
```

Task 3:

```
void startTimer();
void TC3_Handler();

bool isYellowLEDOn = false;
bool isBlueLEDOn = false;
uint16_t overflowCount = 0;

void setup() {
  SerialUSB.begin(9600);
  pinMode(PIN_LED_13, OUTPUT);  // Blue LED
  pinMode(PIN_LED_RXL, OUTPUT); // Yellow LED

  // Configure Generic Clock Generator 2 with a 1024Hz clock
  GCLK->GENDIV.reg = GCLK_GENDIV_ID(2) | GCLK_GENDIV_DIV(46875); // 48MHz / 46875 = 1024Hz
  while (GCLK->STATUS.bit.SYNCBUSY);

  GCLK->GENCTRL.reg = GCLK_GENCTRL_ID(2) | GCLK_GENCTRL_SRC_DFLL48M | GCLK_GENCTRL_GENEN;
  while (GCLK->STATUS.bit.SYNCBUSY);

  startTimer();
}

void loop() {}

void startTimer() {
  // Disable the timer
  TC3->COUNT8.CTRLA.bit.ENABLE = 0;
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);

  // Configure the clock source to use Generic Clock Generator 2
  REG_GCLK_CLKCTRL = (uint16_t) (GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK2 | GCLK_CLKCTRL_ID_T
  while (GCLK->STATUS.bit.SYNCBUSY);

  // Use the 8-bit timer in Normal Frequency Operation mode with a prescaler of 256
  TC3->COUNT8.CTRLA.reg = TC_CTRLA_MODE_COUNT8 | TC_CTRLA_WAVEGEN_NFRQ | TC_CTRLA_PRESCALER_DIV2
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);

  // Enable the overflow interrupt
  TC3->COUNT8.INTENSET.reg = TC_INTENSET_OVF;

  // Enable the IRQ for the timer
  NVIC_EnableIRQ(TC3_IRQn);

  // Enable the timer
  TC3->COUNT8.CTRLA.bit.ENABLE = 1;
  while (TC3->COUNT8.STATUS.bit.SYNCBUSY);
}
```

```
void TC3_Handler() {
  // Check for the overflow interrupt
  if (TC3->COUNT8.INTFLAG.bit.OVF) {
    TC3->COUNT8.INTFLAG.bit.OVF = 1;  // Clear the overflow interrupt flag

    overflowCount++;

    // Toggle Yellow LED every 2 overflows (approx. 1 Hz)
    if (overflowCount % 16 == 0) {
      digitalWrite(PIN_LED_RXL, isYellowLEDOn);
      SerialUSB.println(isYellowLEDOn ? "Yellow LED is on" : "Yellow LED is off");
      isYellowLEDOn = !isYellowLEDOn;
    }

    // Toggle Blue LED every 2 overflows (approx. 0.5 Hz)
    if (overflowCount % 8 == 0) {
      digitalWrite(PIN_LED_13, isBlueLEDOn);
      SerialUSB.println(isBlueLEDOn ? "Blue LED is on" : "Blue LED is off");
      isBlueLEDOn = !isBlueLEDOn;
    }
  }
}
```

Task 4:

```
// LED definitions in the datasheet
// D13 (PIN_LED_13): Blue
// TX (PIN_LED_TXL): Green
// RX (PIN_LED_RXL): Yellow

// LED definitions in the datasheet
// D13 (PIN_LED_13): Blue
// TX (PIN_LED_TXL): Green
// RX (PIN_LED_RXL): Yellow

#define CPU_HZ 48000000
#define TIMER_PRESCALER_DIV 1024
void startTimer(int frequencyHz);
void setTimerFrequency(int frequencyHz);
void TC3_Handler();
bool isBlueLEDOn = false;
bool isYellowLEDOn = false;

unsigned long blueLEDTime = 0;
unsigned long yellowLEDTime = 0;
unsigned long currentTime = 0;


void setup()
{
SerialUSB.begin(9600);
// while(!SerialUSB);
pinMode(PIN_LED_13, OUTPUT);
startTimer(20);
}


void loop() {}

void setTimerFrequency(int frequencyHz) {

int compareValue = (CPU_HZ / (TIMER_PRESCALER_DIV * frequencyHz)) - 1;
TcCount16* TC = (TcCount16*) TC3;
// Make sure the count is in a proportional position to where it was
// to prevent any jitter or disconnect when changing the compare value.
TC->COUNT.reg = map(TC->COUNT.reg, 0, TC->CC[0].reg, 0, compareValue);
TC->CC[0].reg = compareValue;
SerialUSB.println(TC->COUNT.reg);
SerialUSB.println(TC->CC[0].reg);
while (TC->STATUS.bit.SYNCBUSY == 1);
}
```

```
void startTimer(int frequencyHz)
{
  REG_GCLK_CLKCTRL = (uint16_t) (GCLK_CLKCTRL_CLKEN |
                              GCLK_CLKCTRL_GEN_GCLK0 |
                              GCLK_CLKCTRL_ID_TCC2_TC3) ; //set the clock controller, set clock

  while ( GCLK->STATUS.bit.SYNCBUSY == 1 ); // wait for sync

  TcCount16* TC = (TcCount16*) TC3;  // efficient way to do things, cast TC3 to TC.

  // to configure a enable protected register, like WDT we need to disable it first.
  TC->CTRLA.reg &= ~TC_CTRLA_ENABLE; //Disable timer
  while (TC->STATUS.bit.SYNCBUSY == 1); // wait for sync

  // Use the 16-bit timer
  TC->CTRLA.reg |= TC_CTRLA_MODE_COUNT16;
  while (TC->STATUS.bit.SYNCBUSY == 1); // wait for sync

  // Use match mode so that the timer counter resets when the count matches the compare register
  TC->CTRLA.reg |= TC_CTRLA_WAVEGEN_MFRQ;
  while (TC->STATUS.bit.SYNCBUSY == 1); // wait for sync

  // Set prescaler to 1024
  TC->CTRLA.reg |= TC_CTRLA_PRESCALER_DIV1024;
  while (TC->STATUS.bit.SYNCBUSY == 1); // wait for sync

  setTimerFrequency(frequencyHz);

  // Enable the compare interrupt at high it is 0,
  TC->INTENSET.reg = 0;
  TC->INTENSET.bit.MC0 = 1;

  NVIC_EnableIRQ(TC3_IRQn);

  // Enable the control register of the TC
  TC->CTRLA.reg |= TC_CTRLA_ENABLE;

  while (TC->STATUS.bit.SYNCBUSY == 1); // wait for sync
}


// this is the interrupt handler

void TC3_Handler() {
  TcCount16* TC = (TcCount16*) TC3;
  if (TC->INTFLAG.bit.MC0 == 1) {
    TC->INTFLAG.bit.MC0 = 1;

    currentTime += 100; // 100 ms has passed

    // Handle Blue LED
```

```
    if (currentTime - blueLEDTime >= 750) { // 0.5 seconds
      isBlueLEDOn = !isBlueLEDOn;
      digitalWrite(PIN_LED_13, isBlueLEDOn);
      blueLEDTime = currentTime;
      SerialUSB.println(isBlueLEDOn ? "Blue LED is on" : "Blue LED is off");
    }

    // Handle Yellow LED
    if (currentTime - yellowLEDTime >= 1000) { // 1 second
      isYellowLEDOn = !isYellowLEDOn;
      digitalWrite(PIN_LED_RXL, isYellowLEDOn);
      yellowLEDTime = currentTime;
      SerialUSB.println(isYellowLEDOn ? "Yellow LED is on" : "Yellow LED is off");
    }
  }
}
```

References:

The refernces are as follows:

1. Arduino Official Documentation
2. Adafruit Learning Resource
3. Noergaard, Tammy. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers.* Newnes, 2012.
4. Davies, John H. *MSP430 Microcontroller Basics.* Elsevier, 2008.