Title page: EEDG/CE 6302 Lab Report 8
CE6302, Embedded Systems, 302 Laboratory - 83204
Lab Topic: MSP432: UART Communication between two MSP432 boards
Xiongtao Zhang, xxz240008
Lab partner name: Yuyang Hsieh
Group Number 4
Instructor name: Tooraj Nikoubin
TA name: Seyed Saeed
Date: 10/20/2024

1. Objective:

Purpose: In lab 8, the purpose is to explore UART (Universal Asynchronous Receiver/Transmitter) communication and learn how to configure UART communicatIon on MSP432 microcontrollers. UART is a widely used serial communication protocol for transmitting and receiving data between devices. To establish a valid UART connection between transmitter and receiver, configuration for both devices is needed. After that, a demo of sending a character from the transmitter and turning on the corresponding LED based on the receiving character in the receiver is performed to validate the UART connection.

Methods used: UART using serial communication to transmit data. The data frame consists of start bit, data bits, parity bit(optional), and stop bit. Using two register to configure the baud rate(UCAxBRW and UCAxMCTLW), UCAxBRW stores the prescaler value and UCAxMCTLW is the modulator with UCOS16, UCBRSx, and UCBRFx.

Results: After calculating and configuring the value of UCBRx, UCBRFx, and UCBRSx, a desired baud rate is set on both transmitter and receiver in channel A2. And in the transmitter, the channel A0 is also set for receiving characters from the computer. When the transmitter connects to the computer and type character r, g, b in CCS terminal, the red, green, and blue LED light in the receiver will switch on respectively.

Software used: Code Composer Studio (CCS)

Major Conclusions: In the lab, by configuring the Baud rate generator, UART channel A2 is established between transmitter and receiver, and another UART channel A0 is established between transmitter and computer. By type characters in the CCS terminal in the computer, the receiver toggles its LED with a different color correspondingly.

2. Introduction:

2.1 Hardware and Software Background Information
Code Composer Studio is an integrated development environment (IDE) for TI's microcontrollers and processors. It comprises a suite of tools used to develop and debug embedded applications. Code Composer Studio includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler and many other features. Code Composer Studio combines the advantages of the Eclipse® and Theia frameworks with advanced capabilities from TI resulting in a compelling feature-rich environment. The cloud-based Code Composer Studio enables development in the cloud without the need to download and install large amounts of software.

2.2 Purpose of the Experiment

The purpose of this project is to explore and implement UART (Universal Asynchronous Receiver/Transmitter) communication on MSP432 microcontrollers. By configuring UART communication between two MSP432 devices, the project demonstrates how to establish two-way communication between a transmitter and a receiver. The primary goal is to transmit characters from the transmitter side, which are then received by the receiver side to toggle and display different LED colors. This helps in understanding how UART works for serial data transmission and reception, along with hands-on experience in configuring UART on embedded systems like the MSP432.

2.3 Summary of the Experiment

In lab 8, UART connections based on modules A0 and A2 are established. With configuring the Baud rate generator(using SMCLK, BRW=19, set BRF to 9, enable oversampling), the desired baud rate is set By type ‘r’, ‘g’, ‘b’, the character is sent from computer to transmitter, and then sent from transmitter to receiver. By receiving the character in the receiver, it toggles the LED with corresponding color.

2.4 Findings of the experiment

When setting the BRF and BRS bits inside the MCTLW register, a corresponding offset is needed. And when Oversampling Mode (OS16) is on, the clock source frequency is divided by 16 to determine the actual baud rate.

3. Explanation:

3.1 Experiment procedure

This lab exercise focuses on implementing UART communication between two MSP432P401R microcontrollers using Code Composer Studio. The procedure involves creating separate projects for a transmitter and a receiver. For the transmitter, EUSCI_A0 is configured to receive characters from a keyboard, while EUSCI_A2 is set up to transmit these characters to the receiver. Both UART modules are configured with SMCLK as the clock source, and the baud rate is set to 9600 using appropriate BRW and modulation control word calculations. The receiver project configured EUSCI_A2 to receive characters from the transmitter and implemented logic to toggle LED2 based on the received character's color ('r' for red, 'g' for green, 'b' for blue). Error handling is included to blink LED1 twice if an unexpected character is received.

The hardware setup involves connecting two MSP432 LaunchPads to a computer via micro-USB cables. After compiling and loading the respective code onto each board, the UART communication is established by cross-connecting the TX and RX pins (P3.2 and P3.3) between the transmitter and receiver boards. This setup allows for practical implementation and testing of UART communication, providing hands-on experience with serial data transmission, interrupt handling, and GPIO control in embedded systems programming.

3.2 Experiment Images for Part 1

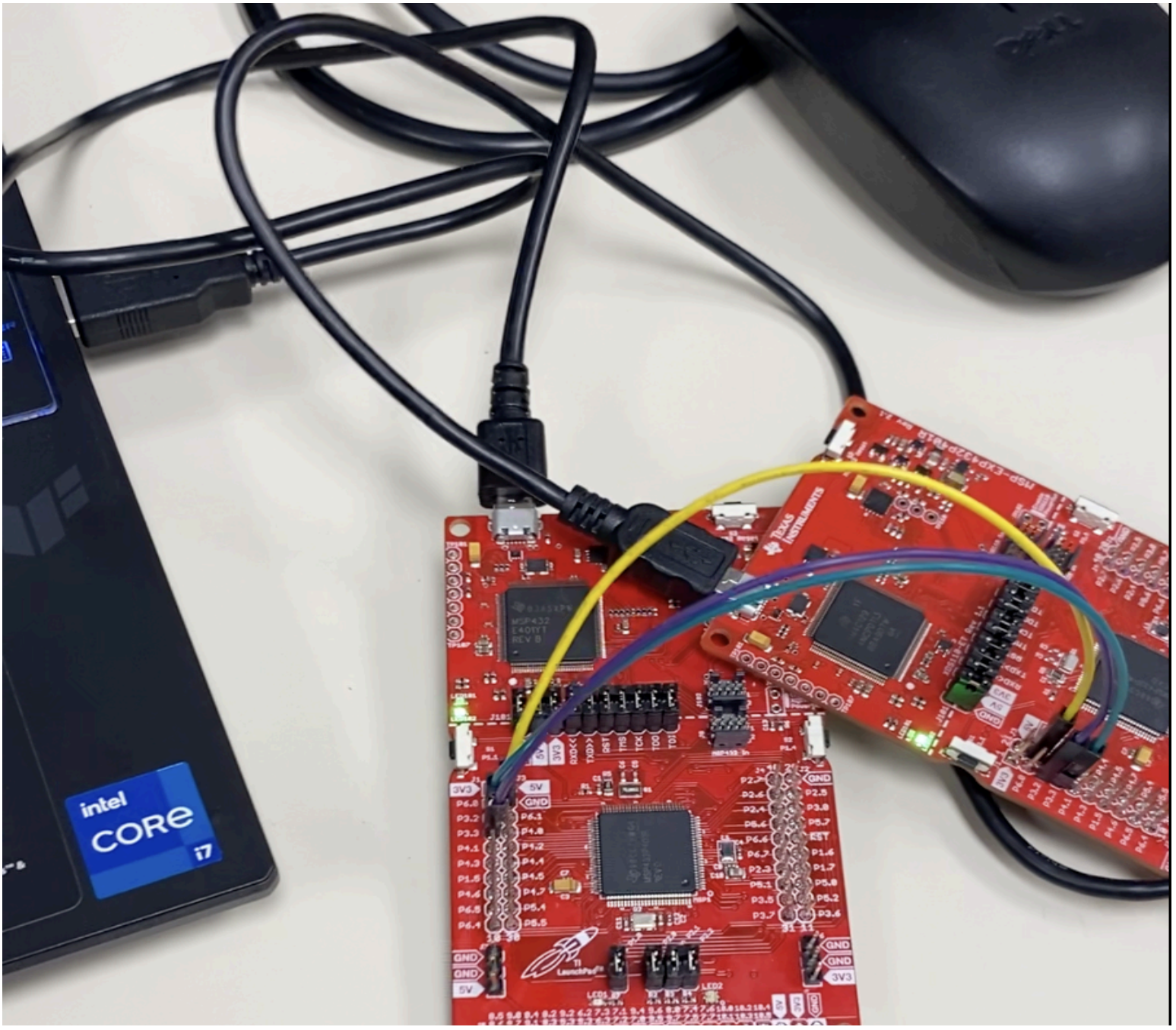

Image 3.2-1: connection between transmitter and computer / transmitter and receiver.

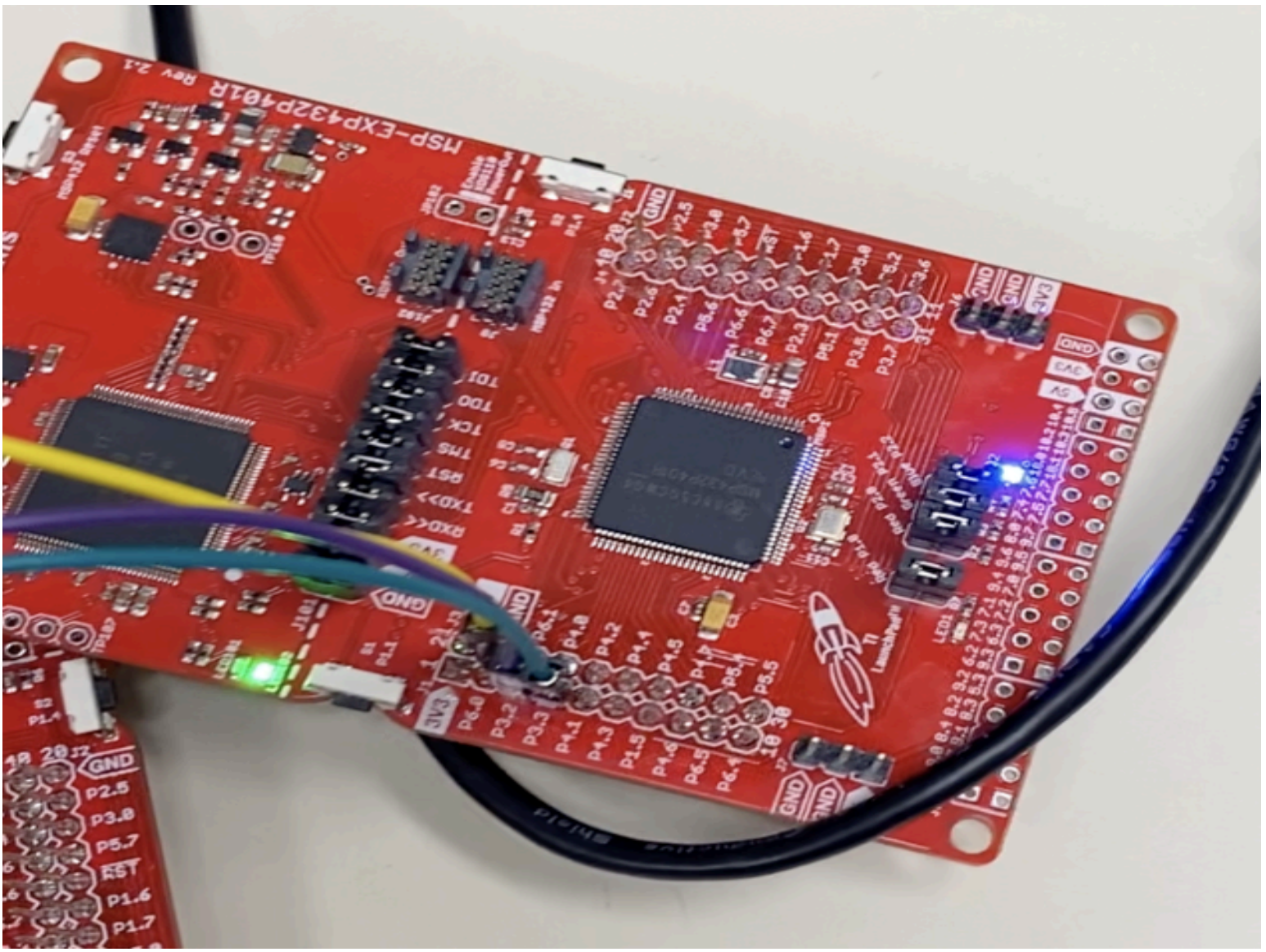

Image 3.2-2: The blue LED on the receiver turns on

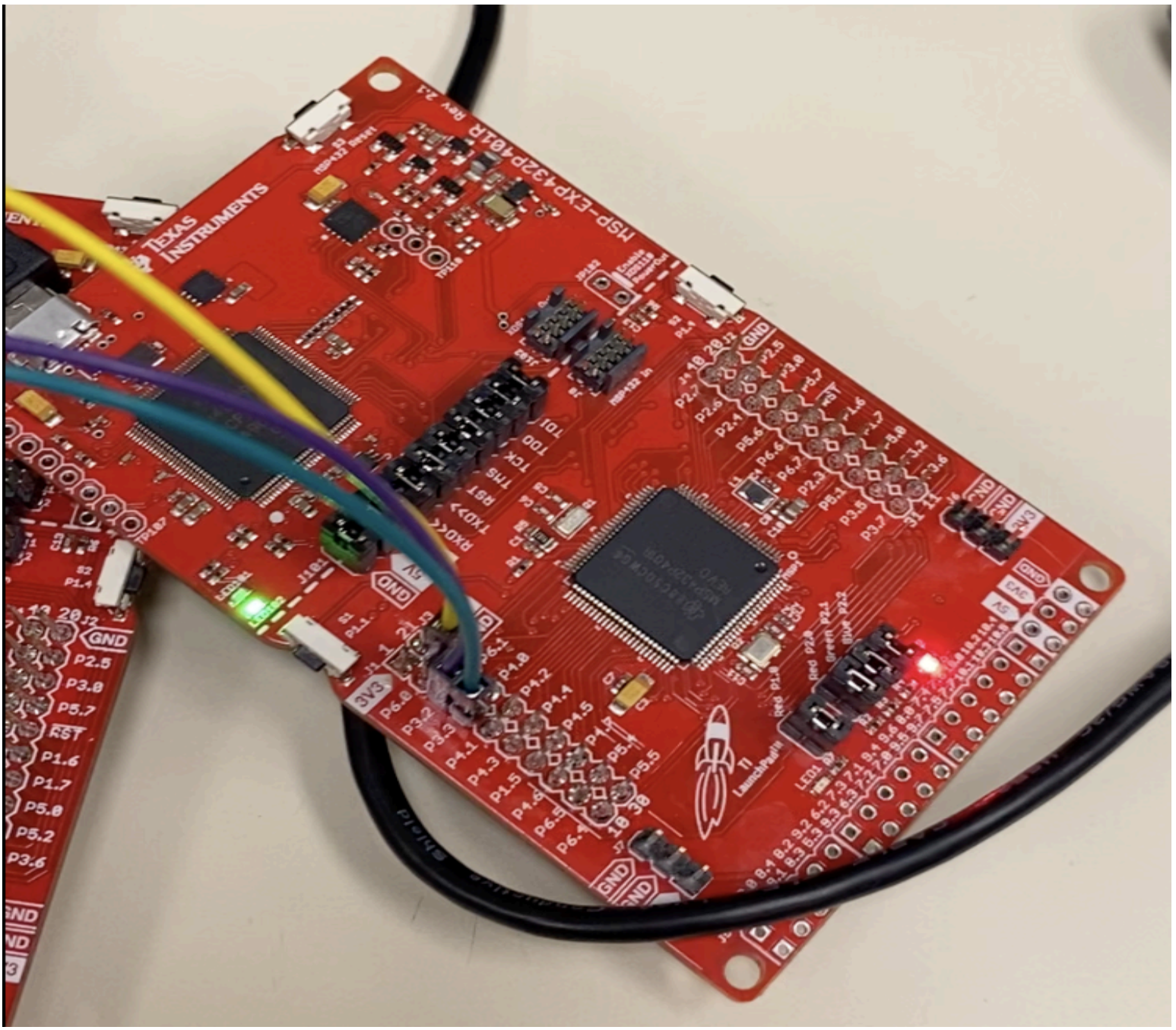

Image 3.2-3: The red LED on the receiver turns on

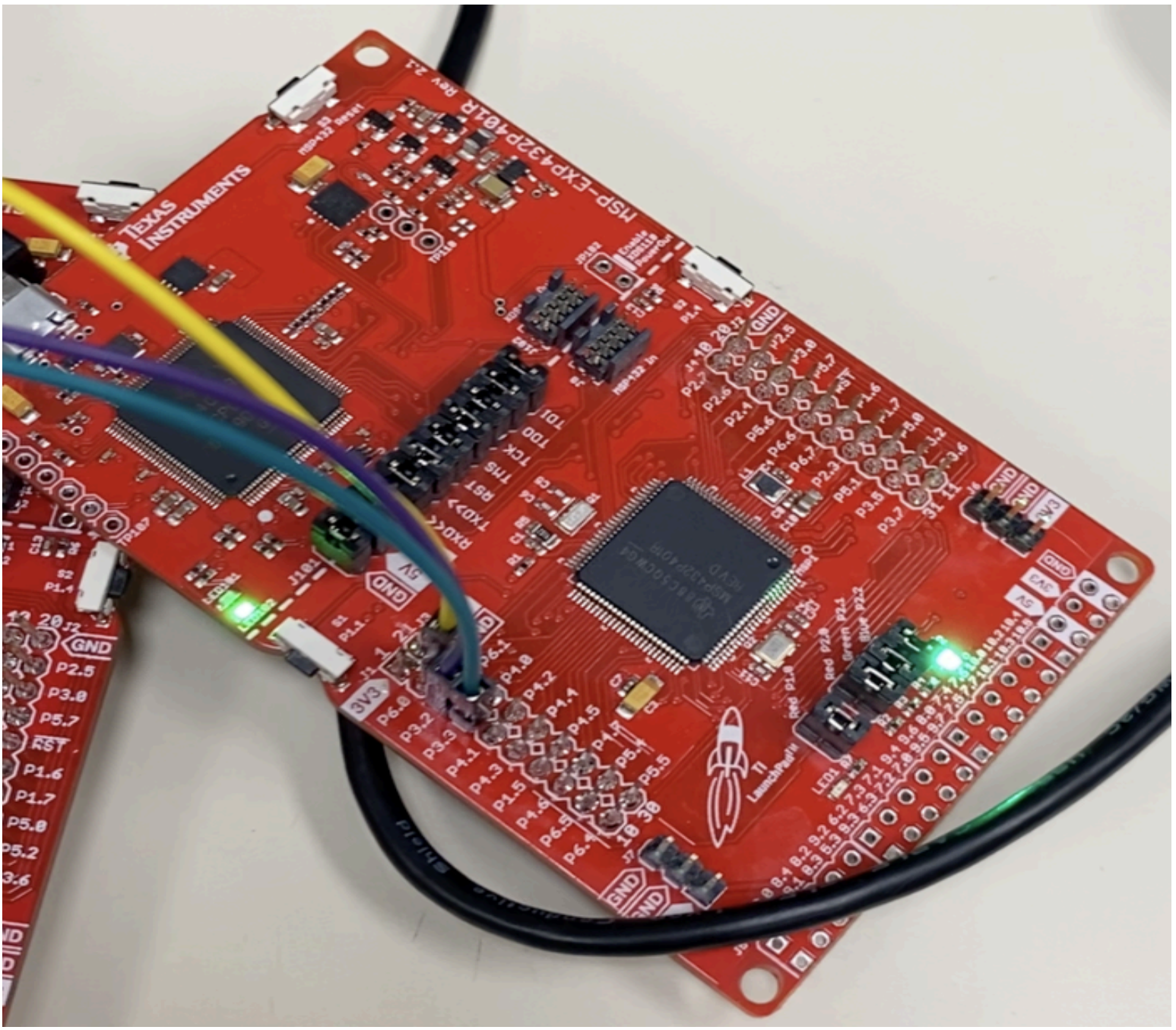

Image 3.2-4: The green LED on the receiver turns on

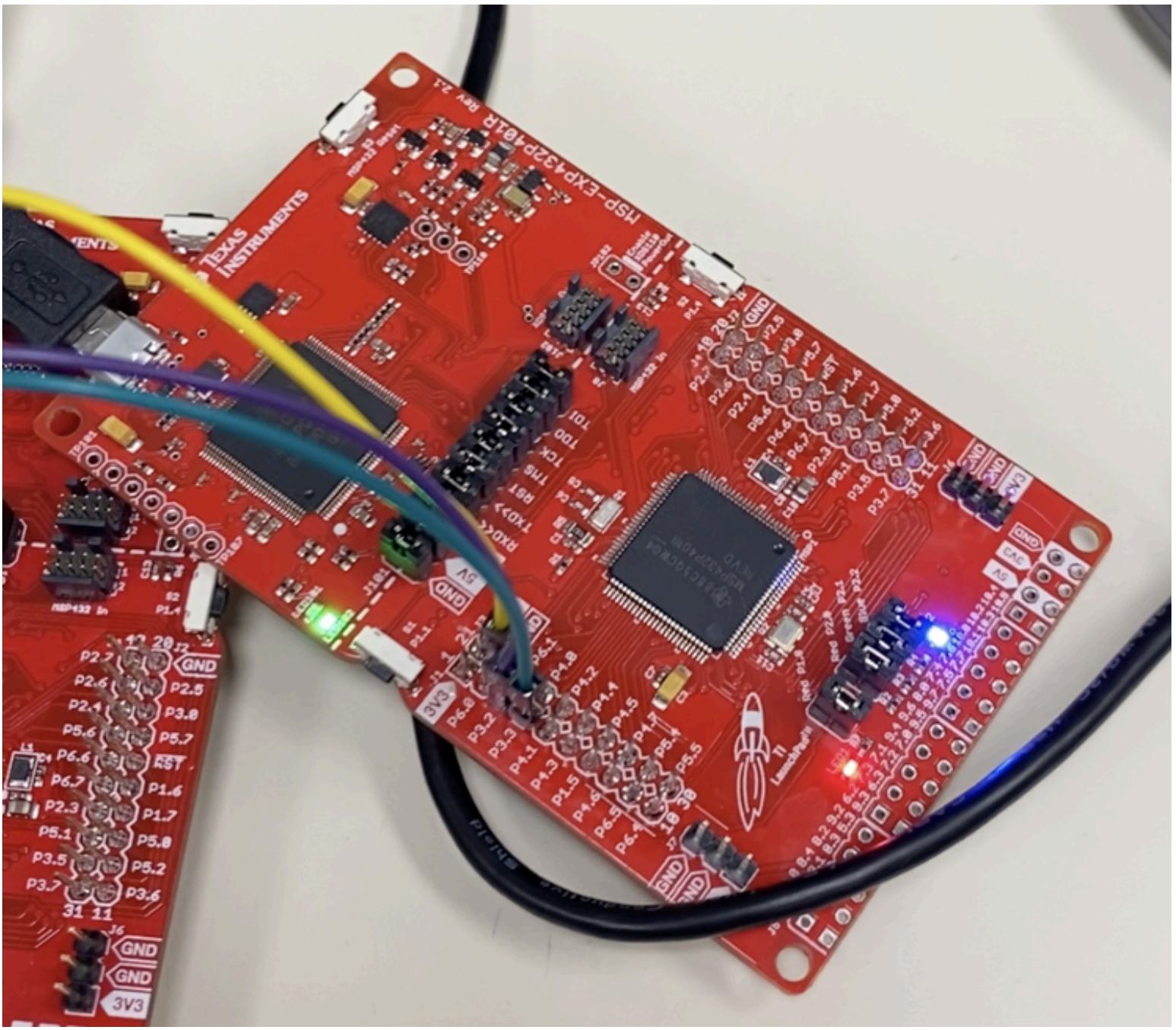

Image 3.2-5: The error LED on the receiver flashes

3.3 Code

```c
#include "msp.h"
#include<stdint.h>
#include<stdbool.h>

void sendString(char *str);
void sendChar(char s);

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer

    // Enable UART0 Pins
    // P1.2->RX
    // P1.3->TX
    P1->SEL0 |= BIT2 | BIT3;
    P1->SEL1 &= ~(BIT2 | BIT3);

    // UART0 Configuration
    EUSCI_A0->CTLW0 = EUSCI_A_CTLW0_SWRST; // Clear previous configuration of UART
    EUSCI_A0->CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK; // Use SMCLK as clock source (3MHz)
    //N = clock source / baud rate = 3000000 / 9600 / 16 = 19
    EUSCI_A0->BRW = 19; // Baud Rate 9600

    //19.53125 - 19 = 0.53125 * 16 = 8.5, round up to 9
    EUSCI_A0->MCTLW = (9 << EUSCI_A_MCTLW_BRF_OFS | EUSCI_A_MCTLW_OS16);

    EUSCI_A0->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Release reset
    EUSCI_A0->IFG &= ~EUSCI_A_IFG_RXIFG; // Clear interrupt flag
    EUSCI_A0->IE |= EUSCI_A_IE_RXIE; // Enable RX interrupt

    // Enable UART2 Pins
    // P3.2->RX
    // P3.3->TX
    P3->SEL0 |= BIT2 | BIT3;
    P3->SEL1 &= ~(BIT2 | BIT3);

    // UART2 Configuration
    EUSCI_A2->CTLW0 = EUSCI_A_CTLW0_SWRST; // Clear previous configuration of UART
    EUSCI_A2->CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK; // Use SMCLK as clock source
    EUSCI_A2->BRW = 19; // Baud Rate 9600

    //19.53125 - 19 = 0.53125 * 16 = 8.5, round up to 9
    EUSCI_A2->MCTLW = (9 << EUSCI_A_MCTLW_BRF_OFS | EUSCI_A_MCTLW_OS16);
    EUSCI_A2->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Release reset

    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG; // Clear interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE; // Enable RX interrupt

    // Enable NVIC for UART0
    NVIC->ISER[0] = 1 << (EUSCIA0_IRQn & 31);
    // Enable global interrupts
    __enable_irq();
```

```
    sendString("Enter r for red, g for green, b for blue!\r\n"); // send message

    while (1)
    {
        // do nothing
    }
}

void EUSCIA0_IRQHandler(void)
{
    if (EUSCI_A0->IFG & EUSCI_A_IFG_RXIFG) // receive interrupt
    {
        char c = EUSCI_A0->RXBUF; // store data into character buffer, and clear flag
        sendString(&c);
        sendChar(c);              // display character through the SERIAL port
    }
}

void sendString(char *str)
{
    int i = 0;
    while (str[i] != '\0')
    {
        while (!(EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG)); // Wait until TXBUF is empty
        EUSCI_A0->TXBUF = str[i]; // Send character through buffer
        i++;
    }
}

void sendChar(char s)
{
    while (!(EUSCI_A2->IFG & EUSCI_A_IFG_TXIFG)); // Wait until TXBUF is empty
        EUSCI_A2->TXBUF = s; // Send character through buffer
}
```

Image 3.3-1: code for transmitter

```c
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer

    // Error LEDs
    P1->DIR |= BIT0;    // set BIT0 as OUTPUT
    P1->OUT &= ~(BIT0); // set BIT0 as LOW
    P1->SEL0 &= ~(BIT0);
    P1->SEL1 &= ~(BIT0);

    // RGB LEDs
    P2->DIR |= BIT0 | BIT1 | BIT2;    // set BIT0,1,2 as OUTPUTs
    P2->OUT &= ~(BIT0 | BIT1 | BIT2); // set BIT0,1,2 as LOW
    P2->SEL0 &= ~(BIT0 | BIT1 | BIT2);
    P2->SEL1 &= ~(BIT0 | BIT1 | BIT2);

    // Enable UART2 Pins (P3.2->RX, P3.3->TX)
    P3->SEL0 |= BIT2 | BIT3;
    P3->SEL1 &= ~(BIT2 | BIT3);

    // UART2 Configuration
    EUSCI_A2->CTLW0 = EUSCI_A_CTLW0_SWRST; // Clear previous configuration of UART
    EUSCI_A2->CTLW0 |= EUSCI_A_CTLW0_SSEL__SMCLK; // Use SMCLK as clock source
    EUSCI_A2->BRW = 19; // Baud Rate 9600

    //19.53125 - 19 = 0.53125 * 16 = 8.5, round up to 9
    EUSCI_A2->MCTLW = (9 << EUSCI_A_MCTLW_BRF_OFS | EUSCI_A_MCTLW_OS16);
    EUSCI_A2->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Release reset

    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG; // Clear interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE; // Enable RX interrupt

    // Enable NVIC for UART2
    NVIC->ISER[0] = 1 << (EUSCIA2_IRQn & 31);

    // Enable global interrupts
    __enable_irq();

    while (1); // Infinite loop
}

void EUSCIA2_IRQHandler(void)
{
    if (EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG) // receive interrupt
    {
        char c = EUSCI_A2->RXBUF; // read the received character
        int i, j; // Declare variables outside the loops

        switch (c)
        {
        case 'r':
            P2->OUT &= ~(BIT0); // set BIT0 as LOW
            P2->OUT &= ~(BIT1); // set BIT0 as LOW
            P2->OUT &= ~(BIT2); // set BIT0 as LOW
```

```
57              P2->OUT ^= BIT0; // Toggle Red LED (P2.0)
58              break;
59          case 'g':
60              P2->OUT &= ~(BIT0); // set BIT0 as LOW
61              P2->OUT &= ~(BIT1); // set BIT0 as LOW
62              P2->OUT &= ~(BIT2); // set BIT0 as LOW
63              P2->OUT ^= BIT1; // Toggle Green LED (P2.1)
64
65              break;
66          case 'b':
67              P2->OUT &= ~(BIT0); // set BIT0 as LOW
68              P2->OUT &= ~(BIT1); // set BIT0 as LOW
69              P2->OUT &= ~(BIT2); // set BIT0 as LOW
70              P2->OUT ^= BIT2; // Toggle Blue LED (P2.2)
71
72              break;
73
74
75          default:
76              // Blink Error LED (P1.0) twice
77              for (i = 0; i < 2; i++)
78              {
79                  P1->OUT ^= BIT0; // Toggle Error LED (P1.0)
80                  for (j = 0; j < 100000; j++); // Simple delay
81                  P1->OUT ^= BIT0; // Toggle Error LED off
82                  for (j = 0; j < 100000; j++); // Simple delay
83              }
84           break;
85      }
86
87      EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG; // Clear the receive interrupt flag
88  }
89 }
```

Image 3.3-2: code for receiver

4. Discussions and Conclusions:

4.1 Comparison of your experimental results with the research/theory of the concept with reasoning.

In this lab, the experimental setup involved configuring UART communication on the MSP432 microcontroller, enabling data exchange between a transmitter and a receiver board to control RGB LEDs. Theoretical concepts, such as baud rate, start/stop bits, and parity, were applied practically to ensure accurate data transmission. The baud rate was set to 9600, matching the typical standards in UART communication, and the experiment successfully demonstrated how microcontrollers manage asynchronous serial data transfer without a shared clock. The LEDs toggled based on received data, confirming the successful implementation of UART interrupts and clock configurations. The results aligned with the theoretical understanding of UART, showcasing its reliability and efficiency in real-time communication and hardware control.

4.2 Learnings from the experiment

This lab provided an understanding of UART (Universal Asynchronous Receiver/Transmitter) communication, a widely used protocol for serial data exchange between devices. It involved configuring UART on the MSP432 microcontroller to enable data transmission and reception between two devices. The setup allowed for the transmission of characters from a transmitter board to control the RGB LEDs on a receiver board. The LEDs toggled based on the received data, demonstrating how to manage communication and control hardware components through interrupts and UART configuration. Key concepts included setting up clock configurations, adjusting the baud rate (in this case, 9600), and using software delay loops for timing. This exercise highlighted how microcontrollers communicate and interact with peripheral devices using UART protocols, ensuring reliable and accurate data transfer.

5. References:

Texas Instruments, https://www.ti.com/tool/CCSTUDIO