Indian Institute of Technology Bombay

Microprocessor Lab
EE 337

Lab 9B Part II
April 14, 2025

Mridul Choudhary
23B3933

Contents

1 Board to Board communication using SPI

1.1 Aim of the experiment

The aim of Part 1 in the EE337 Lab 9B project is to establish board-to-board communication using the SPI (Serial Peripheral Interface) protocol. A Master and a Slave microcontroller board are connected through four SPI lines: CLK, MISO, MOSI, and CS. The Master board sends two prime numbers to the Slave. The Slave computes their sum and checks whether it is a prime number. The result, along with the sum, is displayed on the LCDs attached to both boards. The Master receives a signal from the Slave and shows either "PRIME" or "NOT PRIME" on its LCD.

1.2 Design

1. **Hardware Connections:**
    - Connect both the boards using the SPI lines: CLK, MISO, MOSI, and CS.
    - Attach an LCD to both boards for output display.
    - Connect necessary power and ground pins between boards.
2. **Initialization:**
    - Initialize the SPI interface on both Master and Slave boards using **spi_master_init()** and **spi_slave_init()** respectively.
    - Initialize the LCD on both boards using **lcd_init()**.
3. **Execution Flow:**
    - The Master sends two predefined prime numbers ($p1 = 2$, $p2 = 11$) to the Slave using the SPI transmit function **spi_trx()**.
    - The Slave receives the two numbers, adds them, and checks if the result is also a prime using the **is_prime()** function.
    - The result of the prime check is sent back to the Master.
    - The Slave displays the two numbers and their sum on its LCD.
    - The Master displays either "**PRIME**" or "**NOT PRIME**" based on the result.
    - Displays remain indefinitely.

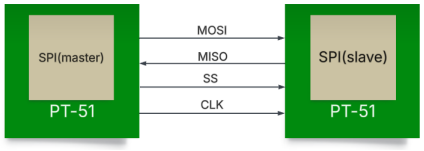

Figure 1: Master - Slave Configuration

1.3 Function Explanations

- **main() in SPI_master.c**
  The main function in the Master board initializes SPI and LCD. It then sends two hard-coded prime numbers to the Slave via SPI. After short delays to allow data transmission, it awaits the response from the Slave. If the response is 1, it displays "PRIME"; if 2, it shows "NOT PRIME"; otherwise, it outputs "ERROR 404". This function demonstrates SPI transmission and LCD feedback effectively.

```
void main(void) {
    unsigned int p1 = 2;
    unsigned int p2 = 11;
    unsigned int result = 0;
    char adc_ip_data_ascii[6]={0,0,0,0,0,'\0'};

    spi_master_init();
    lcd_init();
    lcd_write_string("Master Initializing");

    SS = 0;
    spi_trx(p1);
    SS = 1;

    msdelay(500);

    SS = 0;
    spi_trx(p2);
    SS = 1;

    msdelay(500);

    result = spi_trx(0);
    lcd_cmd(0x01);

    if(result == 1) {
        lcd_write_string("PRIME");
    }
    else if(result == 2) {
        lcd_write_string("NOT PRIME");
    }
    else {
        lcd_write_string("ERROR 404");
    }

    while (1);
}
```

- **main() in SPI_slave.c**
  In the Slave board, the main function initializes the SPI and LCD. It then waits to receive two numbers via SPI from the Master. Upon receiving, it displays the numbers and their sum on its LCD. It uses the helper function is_prime() to determine if the sum is a prime number and sends the result (1 or 2) back to the Master via SPI.

```
void main(void) {
    unsigned int num1, num2, sum, result;
    char adc_ip_data_ascii[6]={0,0,0,0,0,'\0'};

    spi_slave_init();

    lcd_init();
    lcd_write_string("Slave Initializing");

    lcd_cmd(0x01);

    num1 = spi_trx(0x00);
    int_to_string(num1, adc_ip_data_ascii);
    lcd_write_string(adc_ip_data_ascii);

    num2 = spi_trx(0x00);
    lcd_write_char('+');
    int_to_string(num2, adc_ip_data_ascii);
    lcd_write_string(adc_ip_data_ascii);

    sum = num1 + num2;

    result = is_prime(sum);

    lcd_cmd(0xC0);
    lcd_write_string("Sum is ");
    int_to_string(sum, adc_ip_data_ascii);
    lcd_write_string(adc_ip_data_ascii);

    spi_trx(result);

    while (1);
}
```

- **is_prime(unsigned int n) in SPI_slave.c**
  This utility function checks whether a given number is prime. It returns 2 if the number is not prime and 1 if it is. The logic is based on trial division up to the square root of n, which is efficient enough for small inputs and suitable for microcontroller environments.

```
unsigned int is_prime(unsigned int n) {
    unsigned int i;
    if (n < 2) return 2;
    for (i = 2; i * i <= n; ++i) {
    if (n % i == 0) return 2;
    }
    return 1;
}
```

1.4 Output Screenshots

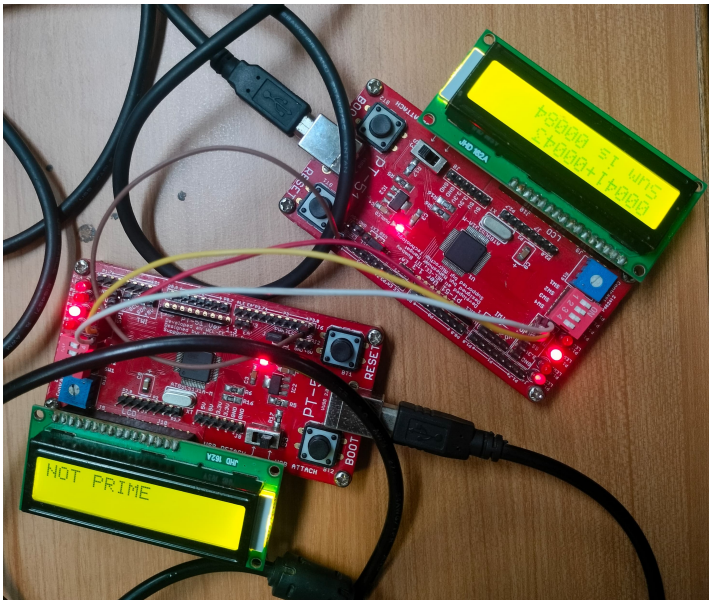

Figure 2: Not Prime Example

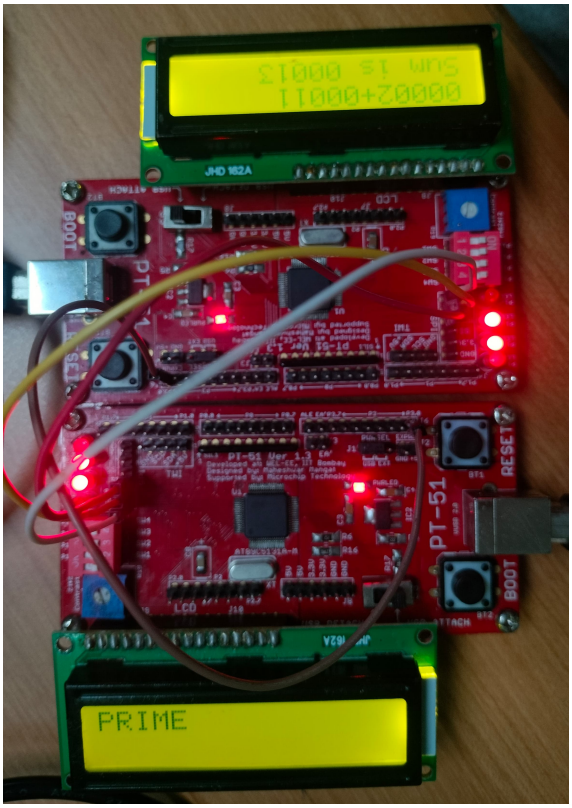

Figure 3: Prime Example

1.5 Conclusion and Inference

This experiment successfully demonstrates SPI-based communication between two microcontroller boards. The Master sends data, the Slave performs processing, and the result is transmitted back, with real-time display on LCDs. The use of SPI provides a practical experience in synchronous communication, and the exercise reinforces concepts in embedded systems, such as peripheral interfacing, basic algorithm implementation, and microcontroller programming.

2 Adding UART and linking ASM file to C

2.1 Aim of the experiment

To extend the SPI-based board-to-board communication system by integrating UART input on the Master side and linking an assembly routine for addition on the Slave side. Specifically:

1. Receive two integers from a PC terminal (RealTerm) via UART on the Master board.
2. Transmit these integers over SPI from Master to Slave.
3. Perform the addition of the received integers in assembly (**add_numbers**) on the Slave and store the result in a shared XDATA variable.
4. Check the primality of the sum in C on the Slave and send the result code back to the Master via SPI.
5. Display "PRIME" or "NOT PRIME" on the PC terminal via UART based on the returned result.

2.2 Design

1. **Hardware Connections:**
   - **SPI:** CLK, MOSI, MISO, CS between Master and Slave.
   - **UART:** TX/RX between Master board and PC (RealTerm).
2. **Software Modules:**
   - **Master (C):**
     - Initialize SPI and UART.
     - Prompt and read two numbers via UART **(receive_char).**
     - Send numbers over SPI **(spi_trx)**.
     - Receive result code via SPI and display via UART
     - (transmit_string).
   - **Slave (C + ASM):**
     - Initialize SPI and LCD (LCD used for debugging).
     - Receive two numbers via SPI **(spi_trx)**.
     - Call assembly routine **add_numbers** to compute **sum**.
     - Check **sum** for primality in C (**is_prime**).
     - Send result code back over SPI.
3. **Memory Layout:**
   - Shared XDATA variables on Slave: **num1**, **num2**, **sum** (16-bit each).

4. **Flow Chart:**

- Master: UART prompt → receive int1, int2.
- Master: SPI send int1, int2; SPI receive result.
- Slave: SPI receive int1, int2 → store in XDATA.
- Slave: **add_numbers()** (ASM) → store **sum**.
- Slave: **is_prime(sum)** → result code (1 = Prime, 2 = Not Prime).
- Slave: SPI send result.
- Master: UART print "PRIME"/"NOT PRIME".

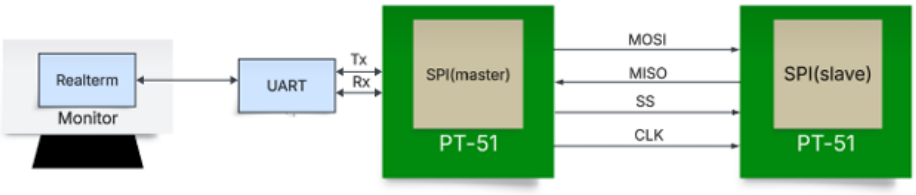

Figure 4: UART - Master - Slave Configuration

2.3 Function Explanations

1. **Master (C)**

- Initialize SPI hardware in Master mode and UART at desired baud.

```
spi_master_init();
uart_init();
```

- Prompt user and accumulate ASCII digits into integer until '.' (ASCII 46) terminator.

```
transmit_string("\r\nEnter Input-1: ");
while(1) { ch = receive_char(); if(ch == '.') break; int1 = int1*10 + (ch -
```

- Pull SS low, transmit each integer via SPI, then deassert SS. Delays ensure proper timing.

```
SS = 0; spi_trx(int1); msdelay(500);
SS = 0; spi_trx(int2); SS = 1;
```

- Read back result code; print corresponding message to terminal.

```
result = spi_trx(0);
if(result == 1) transmit_string("\r\nPRIME\r\n");
else if(result == 2) transmit_string("\r\nNOT PRIME\r\n");
```

2. **Slave (C)**

- Initialize SPI in Slave mode and LCD. Read two 16-bit numbers from Master.

```
spi_slave_init(); lcd_init();
num1 = spi_trx(0x00);
num2 = spi_trx(0x00);
```

- Call external assembly routine to add num1 and num2, result in sum.

```
add_numbers();
```

- Determine primality (returns 1 or 2), send back via SPI.

```
result = is_prime(sum);
spi_trx(result);
```

3. **Addition Routine (Assembly)**

- Uses DPTR and MOVX to access XDATA. Handles 16-bit addition with carry propagation.

```
MOV DPTR, #num1
MOVX A, @DPTR ;
INC DPTR
MOVX A, @DPTR ;


MOV DPTR, #num2
MOVX A, @DPTR
ADD A, R2
JNC NO_CARRY
INC R3
NO_CARRY:
INC DPTR
MOVX A, @DPTR
ADD A, R3
```

```
MOV DPTR, #sum
MOVX @DPTR, R4
INC DPTR
MOVX @DPTR, R5
```

2.4 Output Screenshots

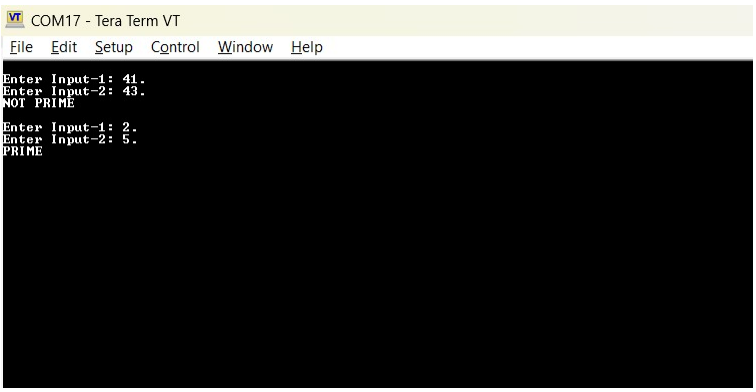

Figure 5: RealTerm Inputs

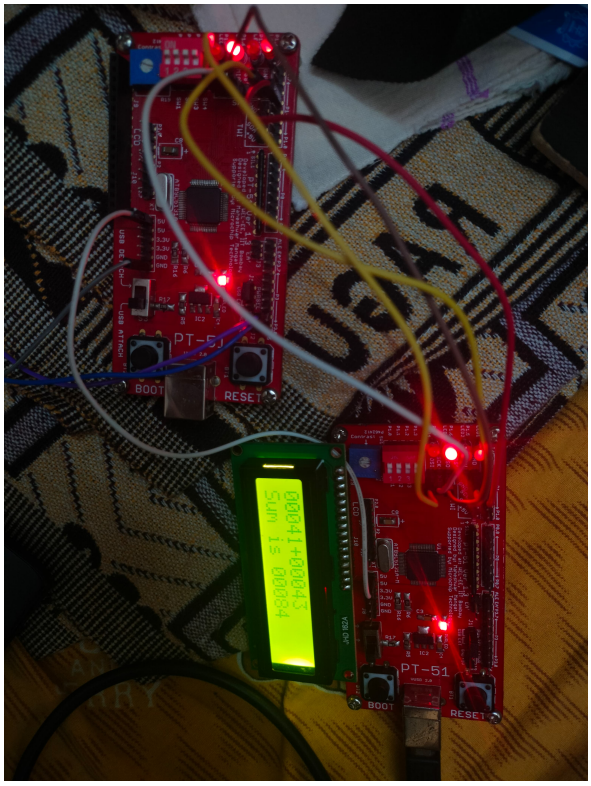

Figure 6: Not Prime Example

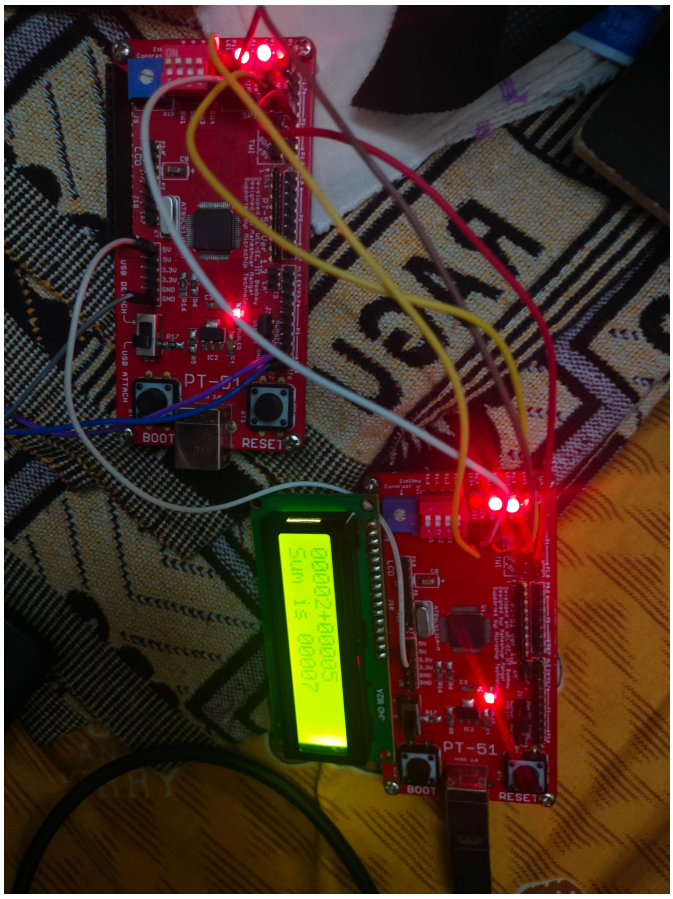

Figure 7: Prime Example

2.5 Conclusion and Inference

The integration of UART with SPI and the use of an assembly subroutine for arithmetic demonstrated:

- **Protocol chaining:** Seamless data flow from UART to SPI and back, enabling flexible I/O.
- **Mixed-language linking:** Successful invocation of an assembly routine from C, handling XDATA variables.
- **Embedded arithmetic:** Correct 16-bit addition with carry management in assembly.
- **Algorithm implementation:** Primality check in C validating results before feedback.

This exercise reinforces multi-protocol interfacing, low-level data handling, and the synergy of C and assembly in resource-constrained microcontrollers.

3 MAC Operation

3.1 Aim of the experiment

To replace the addition operation in the Slave with a Multiply-Accumulate (MAC) operation implemented in assembly, and integrate it into the existing UART–SPI system.

3.2 Design

1. Modify Slave XDATA to include three operands: multiplicand, multiplier, and accumulator.
2. Master reads two operands via UART and sends them over SPI.
3. Slave assembly routine mac_operation performs $sum = sum + (num1 * num2)$.
4. Slave checks the resulting *sum* for primality in C and returns result code via SPI.

3.3 Function Explanations

- **main() in SPI master (C)**
  The main function in the Master board initializes SPI and LCD. It then sends two hard-coded prime numbers to the Slave via SPI. After short delays to allow data transmission, it awaits the response from the Slave. If the response is 1, it displays "PRIME"; if 2, it shows "NOT PRIME"; otherwise, it outputs "ERROR 404". This function demonstrates SPI transmission and LCD feedback effectively.

```
void main(void) {

 spi_master_init();
 uart_init();

 transmit_string("\r\nEnter Input-1: ");
 while(1){
  ch = receive_char();
  if (ch == 46) break;
  int1 = int1 * 10;
  int1 += (ch - 48);
 }

 transmit_string("\r\nEnter Input-2: ");
 while(1){
  ch = receive_char();
  if (ch == 46) break;
  int2 = int2 * 10;
  int2 += (ch - 48);
 }
```

```
transmit_string("\r\nEnter a1: ");
while(1){
 ch = receive_char();
 if (ch == 46) break;
 a1 = a1 * 10;
 a1 += (ch - 48);
}

transmit_string("\r\nEnter a2: ");
while(1){
 ch = receive_char();
 if (ch == 46) break;
 a2 = a2 * 10;
 a2 += (ch - 48);
}

 SS = 0;
 spi_trx(int1);
 SS = 1;

 msdelay(500);

 SS = 0;
 spi_trx(int2);
 SS = 1;

 msdelay(500);

 SS = 0;
 spi_trx(a1);
 SS = 1;
 msdelay(500);

 SS = 0;
 spi_trx(a2);
 SS = 1;
 msdelay(500);

 result = spi_trx(0);
   if (result == 1) {
       transmit_string("\r\nPRIME\r\n");
   } else if(result == 2){
       transmit_string("\r\nNOT PRIME\r\n");
 }
```

```
 while(1);
}
```

- **main() in SPI slave (C)**
  In the Slave board, the main function initializes the SPI and LCD. It then waits to receive two numbers via SPI from the Master. Upon receiving, it displays the numbers and their sum on its LCD. It uses the helper function is_prime() to determine if the sum is a prime number and sends the result (1 or 2) back to the Master via SPI.

```
void main(void) {

 spi_slave_init();
  lcd_init();
 num1=0;
 num2=0;
 sum=0;

 lcd_cmd(0x01);
 lcd_cmd(0x80);
 lcd_write_string("Slave Initialize");


 num1 = spi_trx(0x00);
 lcd_cmd(0x01);
 lcd_cmd(0x80);

 num2 = spi_trx(0x00);

 a1 = spi_trx(0x00);
 a2 = spi_trx(0x00);

 lcd_write_string('*');
 sprintf(buffer, "%d*%d+%d*%d", num1, a1, num2, a2);
 lcd_write_string(buffer);


  mac_operation();

 result = is_prime(sum);
 lcd_cmd(0xC0);

 sprintf(buffer, "Sum is %d", sum);
 lcd_write_string(buffer);

 spi_trx(result);
```

```
 while (1);

}
```

- **mac_operation in SPI slave (Assembly)**
  This utility function checks whether a given number is prime. It returns 2 if the number is not prime and 1 if it is. The logic is based on trial division up to the square root of n, which is efficient enough for small inputs and suitable for microcontroller environments.

```
mac_operation:


            MOV     DPTR, #num1
            MOVX    A, @DPTR

            MOV     DPTR, #a1
            MOVX    A, @DPTR
   MOV B, A
            MUL     AB
            MOV     R0, A



            MOV     DPTR, #num2
            MOVX    A, @DPTR

            MOV     DPTR, #a2
            MOVX    A, @DPTR
   MOV B, A
            MUL     AB
            ADD     A, R0



            MOV     DPTR, #sum
            MOVX    @DPTR, A

            RET

            END
```

3.4 Output Screenshots

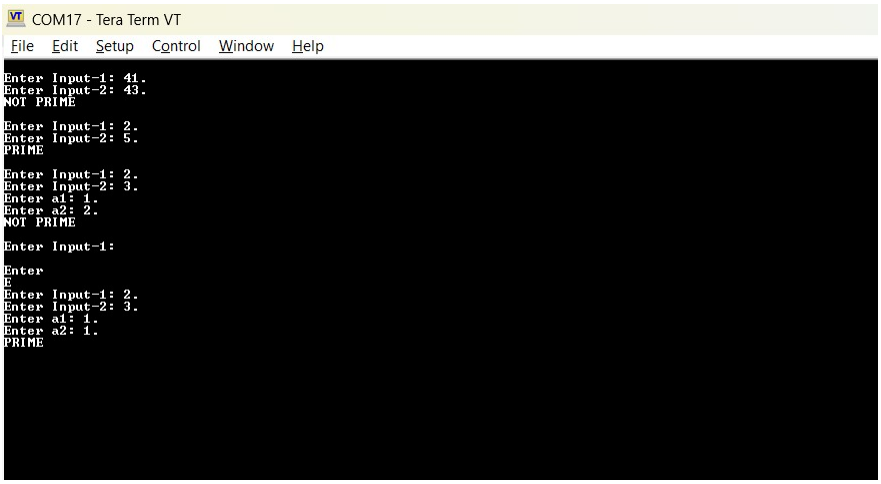

Figure 8: RealTerm Input

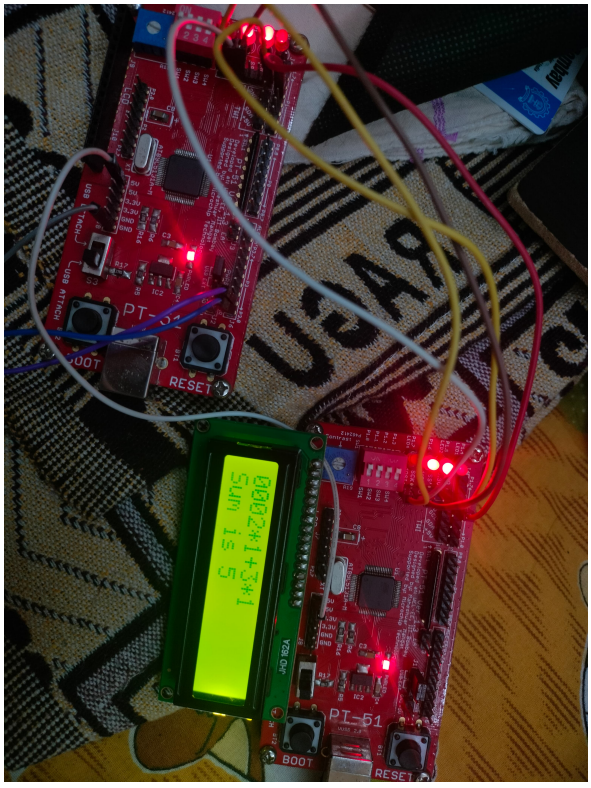

Figure 9: Example

3.5 Conclusion and Inference

Implementing the MAC operation in assembly further demonstrated low-level arithmetic capabilities on the microcontroller. The extended exercise highlights:

- **Assembly proficiency:** Efficient multiply-accumulate in limited instruction set.
- **System flexibility:** Swapping arithmetic routines without altering communication
- **Performance considerations:** Comparing addition vs. MAC timing and code footprint.

This bonus task deepens understanding of DSP-like operations on general-purpose microcontrollers and underscores the power of combining C and assembly for optimized routines.