EE337 Microprocessors Lab
UART to SPI Communication

Digvijay Kumbhar

14 April 2025

Part 1: Board to Board communication using SPI

Objective

To implement board-to-board communication using the SPI protocol. The master board sends two numbers to the slave board, receives a result indicating if their sum is prime, and displays the outcome on the LCD.

Master Code

```
#include <at89c5131.h>
#include "spi.h"
#include "lcd.h"
#include "mcp3008.h"

char lcd_string[6] = {0, 0, 0, 0, 0, '\0'};

void main(void) {
    unsigned int num1 = 2;
    unsigned int num2 = 17;
    unsigned int sum = 0;
    unsigned int checkPrime = 0;

    spi_init();    // Master SPI init
    lcd_init();    // LCD init

    while (1) {
        lcd_cmd(0x80);
        lcd_write_string("Sending nums...");
        msdelay(150);
        spi_trx(num1);        // Send first number
        msdelay(150);
        spi_trx(num2);        // Send second number
        msdelay(150);

        lcd_cmd(0xC0);
        lcd_write_string("Waiting reply...");
        msdelay(150);
        lcd_clear();

        checkPrime = spi_trx(0);  // Receive result from slave
```

```
32
33          if (checkPrime == 1) {
34              lcd_write_string("Prime     ");
35          } else if (checkPrime == 0) {
36              lcd_write_string("Not Prime");
37          } else {
38              lcd_write_string("Error occured");
39          }
40      }
41  }
```

Listing 1: Master SPI Code - Part 1

Working Summary

1. Initializes SPI and LCD peripherals.
2. Sends two hardcoded numbers to the slave.
3. Waits for the result from the slave indicating whether their sum is prime.
4. Displays "Prime" or "Not Prime" or "Error" on the LCD based on the received response.

Slave Side Code Explanation

Objective

The slave board receives two numbers via SPI from the master board. It adds them and checks whether the sum is a prime number. The result (1 if prime, 0 otherwise) is sent back to the master board. It also displays the input numbers and their sum on the LCD.

Prime Checking Function

```
1  bit check_prime(unsigned int n) {
2      unsigned int i;
3      if (n < 2) return 0;
4      for (i = 2; i * i <= n; i++) {
5          if (n % i == 0) return 0;
6      }
7      return 1;
8  }
```

Listing 2: Function to check if number is prime

Explanation: This function checks if a number is prime: - Returns 0 if number is less than 2. - Otherwise, checks for divisibility from 2 up to n. - Returns 1 if it is a prime number.

Main Function

```c
void main(void) {
    unsigned int sum;

    spi_init();     // Initialize SPI
    lcd_init();     // Initialize LCD

    while (1) {
        lcd_cmd(0x80);            // Set cursor to first line
        x = spi_trx(0);           // Receive first number
        y = spi_trx(0);           // Receive second number

        sum = x + y;

        spi_trx(check_prime(sum));  // Send result back to master

        lcd_write_string("X:");
        int_to_string(x, lcd_string);
        lcd_write_string(lcd_string);

        lcd_write_string("Y:");
        int_to_string(y, lcd_string);
        lcd_write_string(lcd_string);

        lcd_cmd(0xC0); // Move to second line
        int_to_string(sum, lcd_string);
        lcd_write_string(lcd_string);
    }
}
```

Listing 3: Slave main function

Working Summary

- Receives two numbers from the master board via SPI.
- Adds the two numbers.
- Checks whether the sum is a prime number using the check_prime() function.
- Sends back the result (1 or 0) to the master.
- Displays the input values and their sum on the LCD screen.

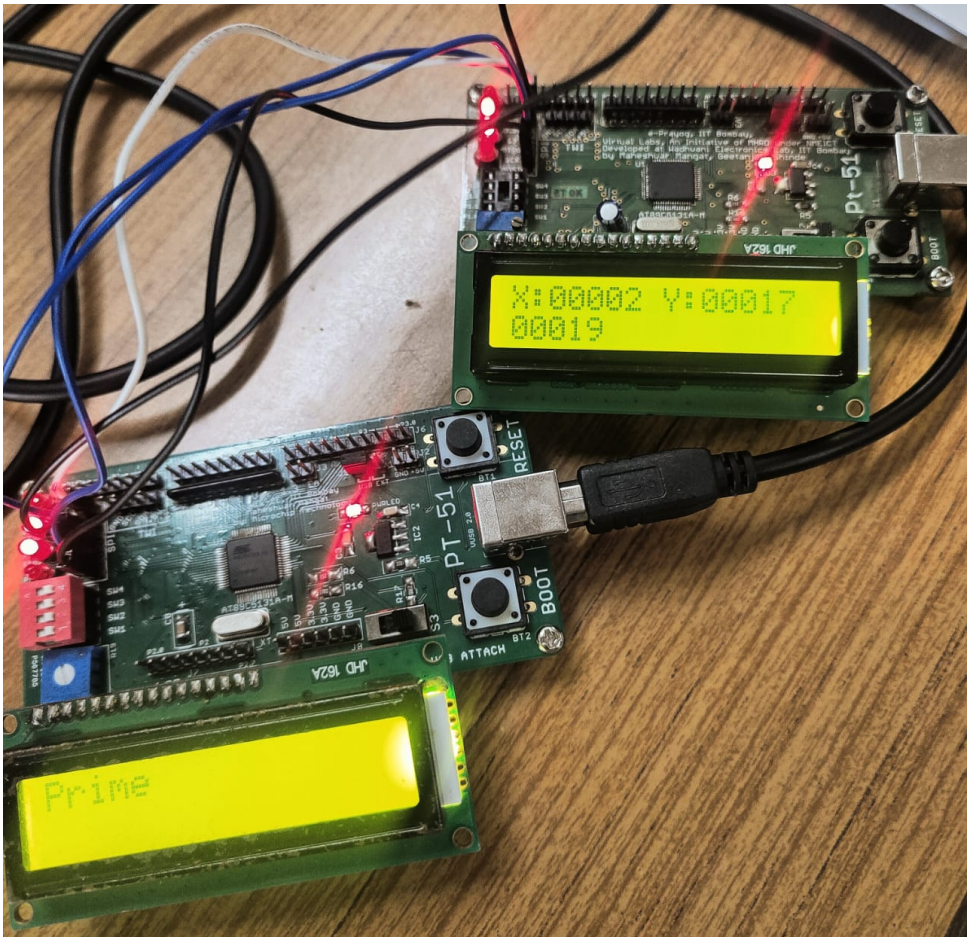

Figure 1

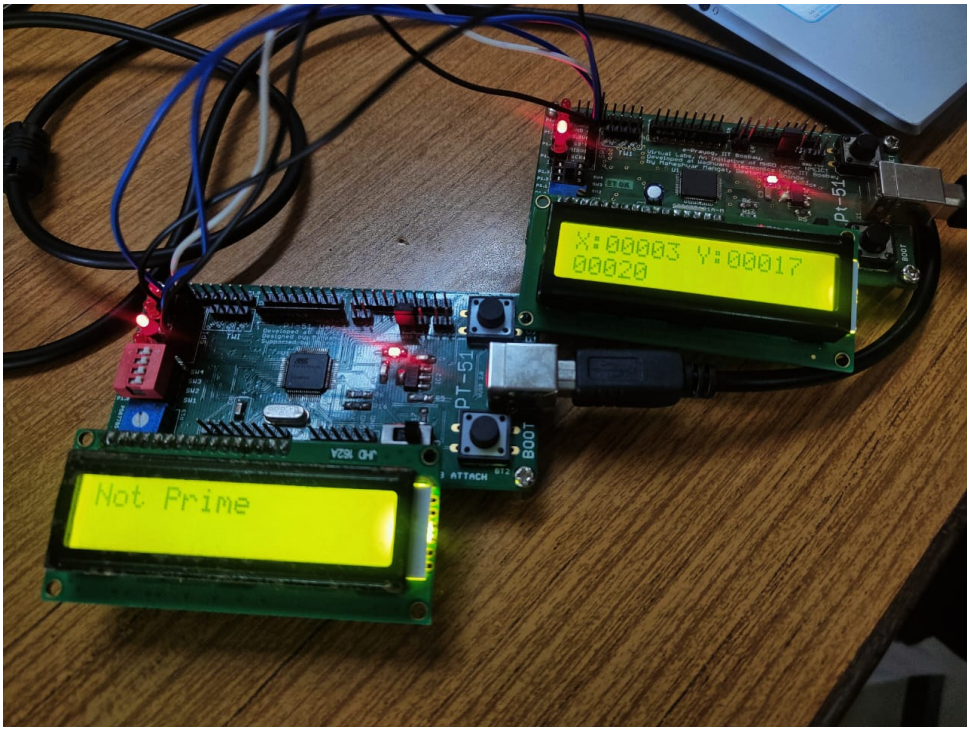

Figure 2

Part 2: UART to SPI Communication

The goal of this part is to implement a UART to SPI communication system using the AT89C5131 microcontroller. The master board receives two numbers from a computer via UART and transmits them to a slave board via SPI. The slave checks whether the sum is a prime number and sends the result back to the master. The master then displays the result on an LCD and transmits it over UART.

Code Explanation

Header Files

- `at89c5131.h` — Register definitions for the microcontroller.
- `spi.h` — Functions for SPI communication.
- `lcd.h` — Functions for LCD operations.
- `mcp3008.h` — (Not directly used here but generally included for ADC via SPI).
- `serial.h` — UART transmit and receive functions.

UART Input Function

The `read_number_uart()` function reads multi-digit input from the UART terminal:

```
unsigned char read_number_uart() {
    unsigned char ch;
    unsigned int num = 0;

    while (1) {
        ch = receive_char();
        if (ch == '\r' || ch == '\n') break;

        if (ch >= '0' && ch <= '9') {
            transmit_char(ch);
            num = num * 10 + (ch - '0');
        }
    }
    return (unsigned char)num;
}
```

Listing 4: Read number via UART

Explanation:

1. Waits for the user to type a character.
2. If the character is a digit ('0' to '9'):
   - It shows the same digit back on the terminal.
   - Converts the character to a number.
   - Adds it to the number being formed.
3. If the user presses Enter, it stops reading and returns the final number.

Main Program

```
void main(void) {
    unsigned char num1 = 0, num2 = 0, checkPrime = 0;

    spi_init();
    lcd_init();
    uart_init();

    lcd_clear();
    lcd_write_string("Enter num1:");
    num1 = read_number_uart();

    lcd_write_string("Enter num2:");
    num2 = read_number_uart();

    lcd_write_string("Sending...");
    spi_trx(num1);
    msdelay(150);
    spi_trx(num2);
    msdelay(150);

    checkPrime = spi_trx(0);

    if (checkPrime == 1) {
        lcd_write_string("Prime");
        transmit_string("Prime\r\n");
    } else if (checkPrime == 0) {
        lcd_write_string("Not Prime");
        transmit_string("Not Prime\r\n");
    } else {
        lcd_write_string("Error");
        transmit_string("Error\r\n");
    }

    while(1);
}
```

Listing 5: Main function overview

Working Summary

1. Initializes SPI, LCD, and UART peripherals.
2. Prompts the user via LCD and UART to enter two numbers.
3. Reads the numbers using UART and echoes them back.
4. Sends the numbers to the slave using SPI.
5. Receives the result (1 = Prime, 0 = Not Prime) from the slave.
6. Displays the result on the LCD and sends it via UART to the Realterm terminal.

Same slave used for this from part 1.

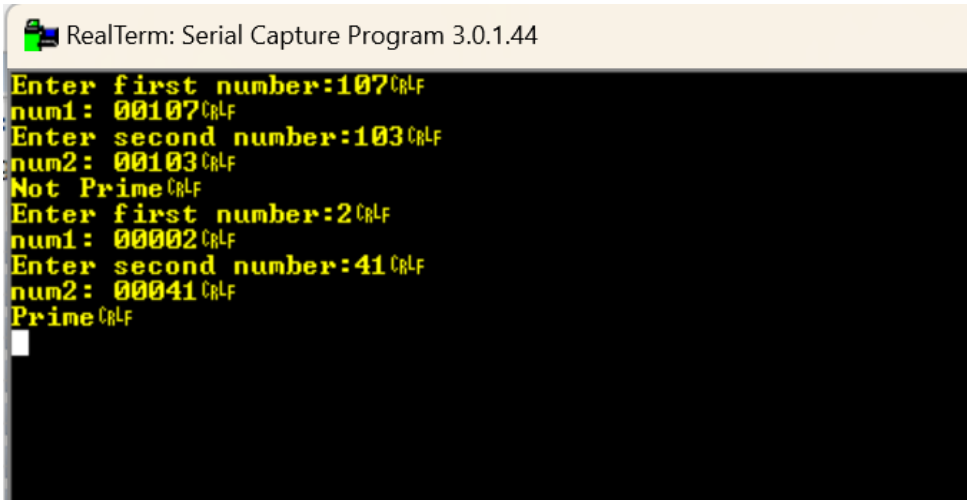

Figure 3

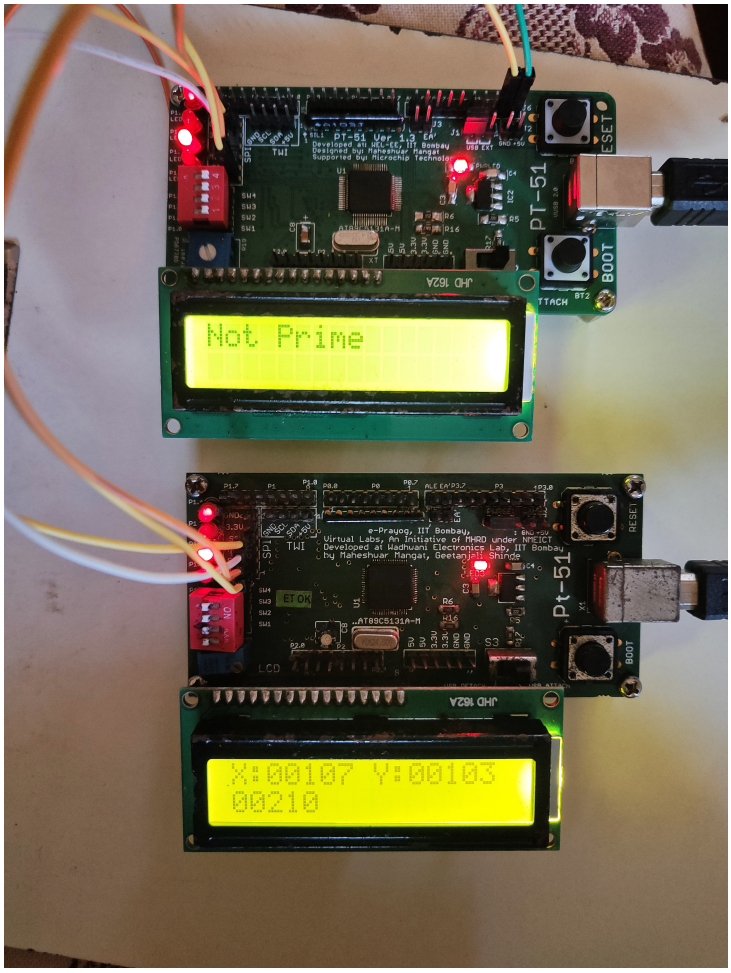

Figure 4

Figure 5

Part 3: Linking asm file with C

Objective

This part demonstrates how to use an Assembly function to perform the addition of two numbers received via SPI in C. The sum is then checked for primality using a C function. The result is sent back to the master via SPI and displayed on the LCD.

Assembly Code: add.a51

This is the Assembly code that adds num1 and num2 and stores the result.

```
; Adds num1 and num2 and stores result in result (all in internal RAM)

PUBLIC ADD_NUMS

EXTRN DATA (num1, num2, result)   ; External variables from C

CSEG AT 100h

ADD_NUMS:
    MOV A, num1       ; Load num1 into accumulator
    ADD A, num2       ; Add num2 to A
    MOV result, A     ; Store sum in result
    RET               ; Return to C

END
```

Listing 6: Assembly: add.a51

C Code with Assembly

```
#include <at89c5131.h>
#include "lcd.h"
#include "spi.h"
#include "serial.h"

char lcd_string[6] = {0, 0, 0, 0, 0, '\0'};

data unsigned char num1 _at_ 0x20;  // Shared using internal RAM
data unsigned char num2 _at_ 0x21;
data unsigned char result _at_ 0x22;

extern void ADD_NUMS(void);  // Declaration of external assembly
   function

// Prime number checker
bit check_prime(unsigned int n) {
    unsigned int i;
    if (n < 2) return 0;
    for (i = 2; i * i <= n; i++) {
        if (n % i == 0) return 0;
    }
    return 1;
}

```

```c
void main(void) {
    unsigned int sum;

    spi_init();
    lcd_init();

    lcd_cmd(0x80);
    lcd_write_string("Receiving...");

    num1 = spi_trx(0);     // Get first number
    num2 = spi_trx(0);     // Get second number

    ADD_NUMS();            // Perform result = num1 + num2 (assembly)

    sum = result;          // Store result in sum

    spi_trx(check_prime(sum));  // Send 1 if prime, else 0

    lcd_clear();
    lcd_write_string("X:");
    int_to_string(num1, lcd_string);
    lcd_write_string(lcd_string);

    lcd_write_string(" Y:");
    int_to_string(num2, lcd_string);
    lcd_write_string(lcd_string);

    lcd_cmd(0xC0);
    int_to_string(result, lcd_string);
    lcd_write_string(lcd_string);

    while (1);  // Infinite loop
}
```

Listing 7: Slave side C code calling assembly

How It Works

- The variables num1, num2, and result are placed in fixed RAM addresses using the data keyword.
- The assembly function ADD_NUMS reads num1 and num2, adds them, and writes the result back into result.
- This result is used by the C code to check if it is a prime number.
- The result is sent back to the master and displayed on the LCD.

The same master code used for this is from part 2.

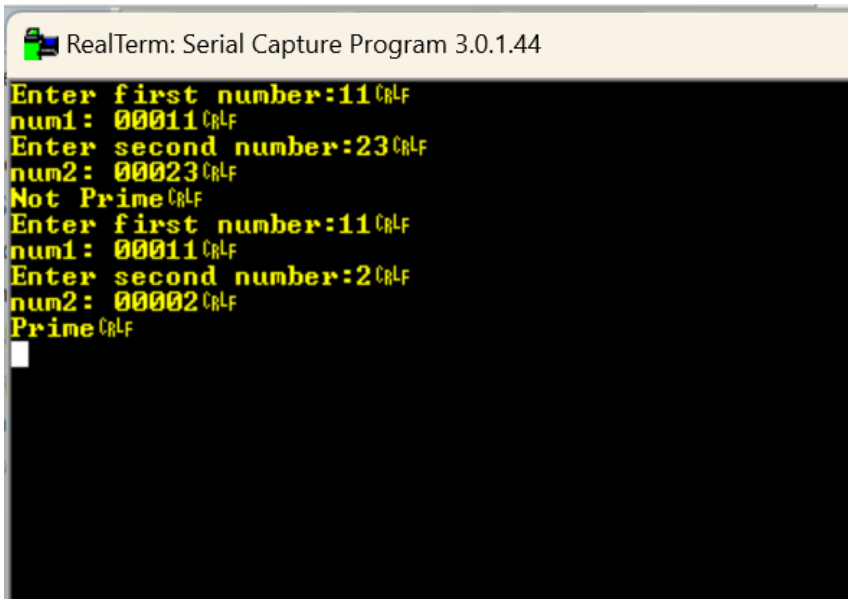

Figure 6

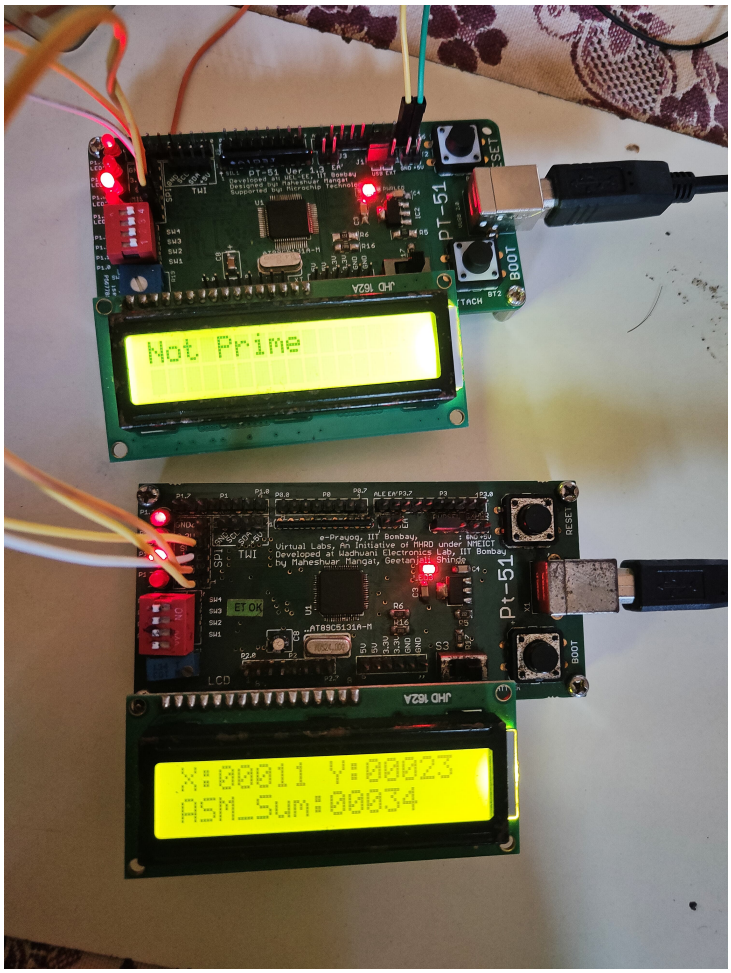

Figure 7

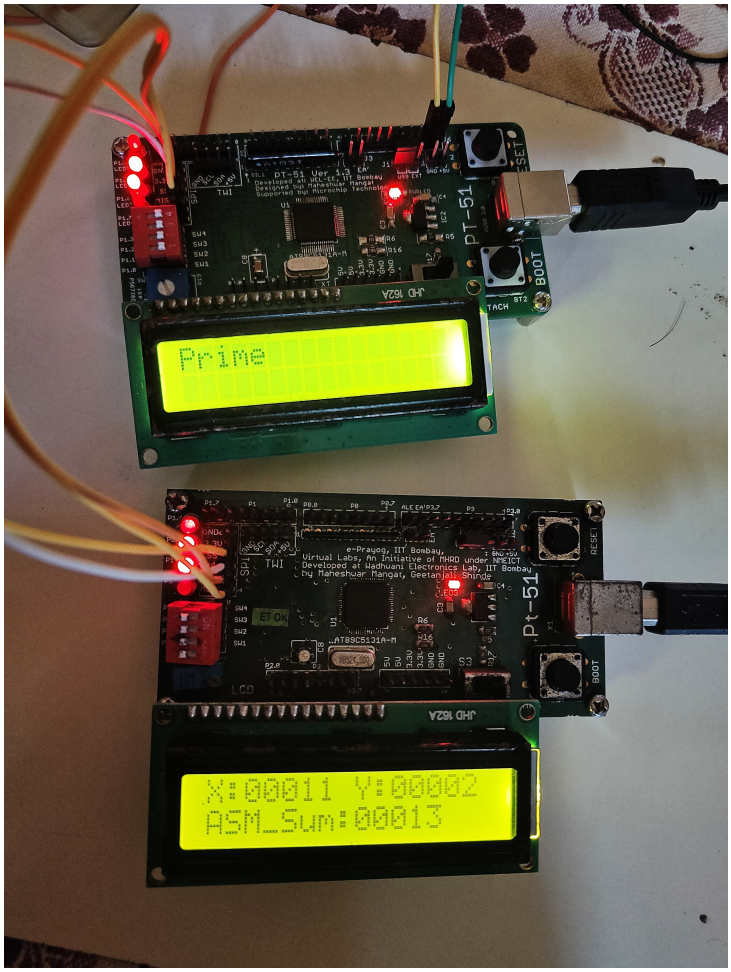

Figure 8

Bonus Part: MAC operation

Objective

To perform a Multiply-Accumulate (MAC) operation of the form:

$$\text{MAC} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

on the slave board using an assembly subroutine. The inputs $a_1, a_2, a_3, b_1, b_2, b_3$ are entered via UART on the master board, transferred over SPI to the slave, where the MAC is computed. The slave sends back whether the MAC result is a prime number or not.

Master Code (C - UART + SPI)

```c
#include <at89c5131.h>
#include "spi.h"
#include "lcd.h"
#include "serial.h"

char lcd_string[6] = { 0, '\0' };

unsigned char read_number_uart() {
    unsigned char ch;
    unsigned int num = 0;

    while (1) {
        ch = receive_char();
        if (ch == '\r' || ch == '\n') break;

        if (ch >= '0' && ch <= '9') {
            transmit_char(ch);
            num = num * 10 + (ch - '0');
        }
    }
    return (unsigned char)num;
}

void main(void) {
    unsigned char a1, a2, a3, b1, b2, b3;
    unsigned char checkPrime = 0;

    spi_init();
    lcd_init();
    uart_init();

    // Read a1, a2, a3 via UART
    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter X1:"); transmit_string("Enter X1: ");
    a1 = read_number_uart(); transmit_string("\r\n");

    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter X2:"); transmit_string("Enter X2: ");
    a2 = read_number_uart(); transmit_string("\r\n");

    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter X3:"); transmit_string("Enter X3: ");
```

```
    a3 = read_number_uart(); transmit_string("\r\n");

    // Read b1, b2, b3 via UART
    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter Y1:"); transmit_string("Enter Y1: ");
    b1 = read_number_uart(); transmit_string("\r\n");

    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter Y2:"); transmit_string("Enter Y2: ");
    b2 = read_number_uart(); transmit_string("\r\n");

    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Enter Y3:"); transmit_string("Enter Y3: ");
    b3 = read_number_uart(); transmit_string("\r\n");

    // Show sending message
    lcd_clear(); lcd_cmd(0x80);
    lcd_write_string("Sending...");

    // Send 6 values to Slave
    spi_trx(a1); msdelay(100);
    spi_trx(a2); msdelay(100);
    spi_trx(a3); msdelay(100);
    spi_trx(b1); msdelay(100);
    spi_trx(b2); msdelay(100);
    spi_trx(b3); msdelay(100);

    // Receive result from Slave
    checkPrime = spi_trx(0);

    // Show result
    lcd_clear(); lcd_cmd(0x80);
    if (checkPrime == 1) {
        lcd_write_string("Result: Prime");
        transmit_string("MAC Result: Prime\r\n");
    } else {
        lcd_write_string("Result:Not Prime");
        transmit_string("MAC Result: Not Prime\r\n");
    }

    while (1);
}
```

Listing 8: Master code to send 6 inputs via UART and receive MAC result from Slave

Assembly Code for MAC Operation (mac.a51)

The following code performs a Multiply-Accumulate (MAC) operation using three pairs of inputs: a_1, a_2, a_3 and b_1, b_2, b_3. Each product is accumulated into a 8-bit result stored across three registers: `mac_lo`, `mac_mid`, and `mac_hi`.

```
PUBLIC MAC_NUMS
EXTRN DATA (a1, a2, a3, b1, b2, b3)
EXTRN DATA (mac_result)

CSEG AT 100h

MAC_NUMS:
    ; Clear accumulator
    MOV A, #00H

    ; First product: a1 * b1
    MOV R0, A           ; Clear sum
    MOV A, a1
    MOV B, b1
    MUL AB
    ADD A, R0           ; Add to sum
    MOV R0, A

    ; Second product: a2 * b2
    MOV A, a2
    MOV B, b2
    MUL AB
    ADD A, R0
    MOV R0, A

    ; Third product: a3 * b3
    MOV A, a3
    MOV B, b3
    MUL AB
    ADD A, R0

    ; Final 8-bit result
    MOV mac_result, A
    RET

END
```

Listing 9: Assembly code to compute MAC = a1*b1 + a2*b2 + a3*b3

Slave Code (C) for MAC Operation

The following code runs on the slave board. It receives 6 values over SPI, performs the MAC operation using an external assembly routine `MAC_NUMS`, checks whether the result is prime, sends the result status back to the master, and displays all values and the MAC result on the LCD.

```c
#include <at89c5131.h>
#include "lcd.h"
#include "spi.h"
#include "serial.h"

char lcd_string[3] = {0};
char lcd_string1[4] = {0};

data unsigned char a1 _at_ 0x70;
data unsigned char a2 _at_ 0x71;
data unsigned char a3 _at_ 0x72;
data unsigned char b1 _at_ 0x73;
data unsigned char b2 _at_ 0x74;
data unsigned char b3 _at_ 0x75;

data unsigned char mac_result _at_ 0x50; // single byte MAC result

extern void MAC_NUMS(void);  // Assembly subroutine

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

void main(void) {
    unsigned int result;

    spi_init();
    lcd_init();

    lcd_cmd(0x80);
    lcd_write_string("Receiving...");

    // Receive 6 numbers (3 x, 3 y)
    a1 = spi_trx(0);
    a2 = spi_trx(0);
    a3 = spi_trx(0);

    b1 = spi_trx(0);
    b2 = spi_trx(0);
    b3 = spi_trx(0);

    // Call MAC routine
    MAC_NUMS();

    // Read 8-bit result
    result = mac_result;
```

```
52
53      // Send 1 if result is prime, else 0
54      spi_trx(check_prime(result));
55
56      lcd_clear();
57      lcd_cmd(0x80);
58
59      // Display a1 a2 a3
60      int_to_string(a1, lcd_string); lcd_write_string(lcd_string);
61      lcd_write_string(" "); int_to_string(a2, lcd_string);
        lcd_write_string(lcd_string);
62      lcd_write_string(" "); int_to_string(a3, lcd_string);
        lcd_write_string(lcd_string);
63
64      lcd_write_string(" ");
65      int_to_string(b1, lcd_string); lcd_write_string(lcd_string);
66      lcd_cmd(0xC0);
67      int_to_string(b2, lcd_string); lcd_write_string(lcd_string);
68      lcd_write_string(" "); int_to_string(b3, lcd_string);
        lcd_write_string(lcd_string);
69
70      // Show MAC result
71      lcd_write_string(" MAC:");
72      int_to_string(result, lcd_string); lcd_write_string(lcd_string);
73
74      while (1);
75  }
```

Listing 10: Slave code for MAC operation and display

Working Summary

- **Input via UART (Master):** User enters six values: `a1, a2, a3, b1, b2, b3` through RealTerm software on a laptop.
- **UART to Master Board:** The Master board receives these values via UART and stores them in variables.
- **Master sends data via SPI:** The Master sends all six values one by one to the Slave board using the SPI protocol.
- **Slave receives data:** The Slave board receives and stores the six incoming values in specific memory addresses.
- **MAC Operation on Slave (Assembly):** The Slave calls the assembly subroutine `MAC_NUMS` to compute:

$$\text{MAC} = a1 \cdot b1 + a2 \cdot b2 + a3 \cdot b3$$

  The result is stored as a 8-bit number across three memory locations: `mac_lo`, `mac_mid`, and `mac_hi`.
- **Prime Check (C function on Slave):** The Slave checks if the MAC result is a prime number using a standard iterative method.

- **Slave sends result via SPI:** The Slave sends the result of the prime check (1 for prime, 0 for not prime) back to the Master via SPI.
- **Display result:**
  - **Slave LCD:** Displays all six input values and the final MAC result.
  - **Master LCD and UART:** Displays whether the MAC result is `Prime` or `Not Prime`.

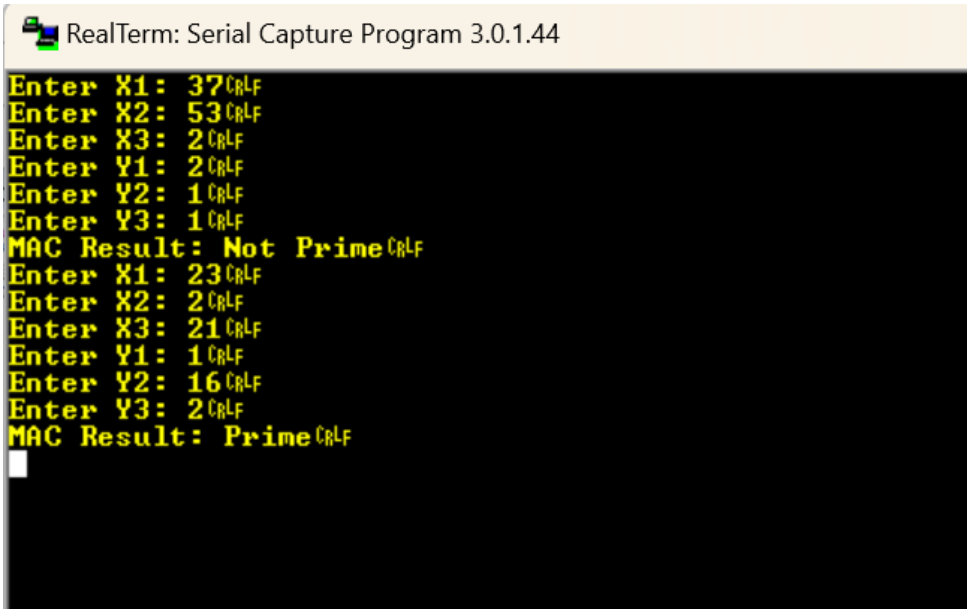

Figure 9

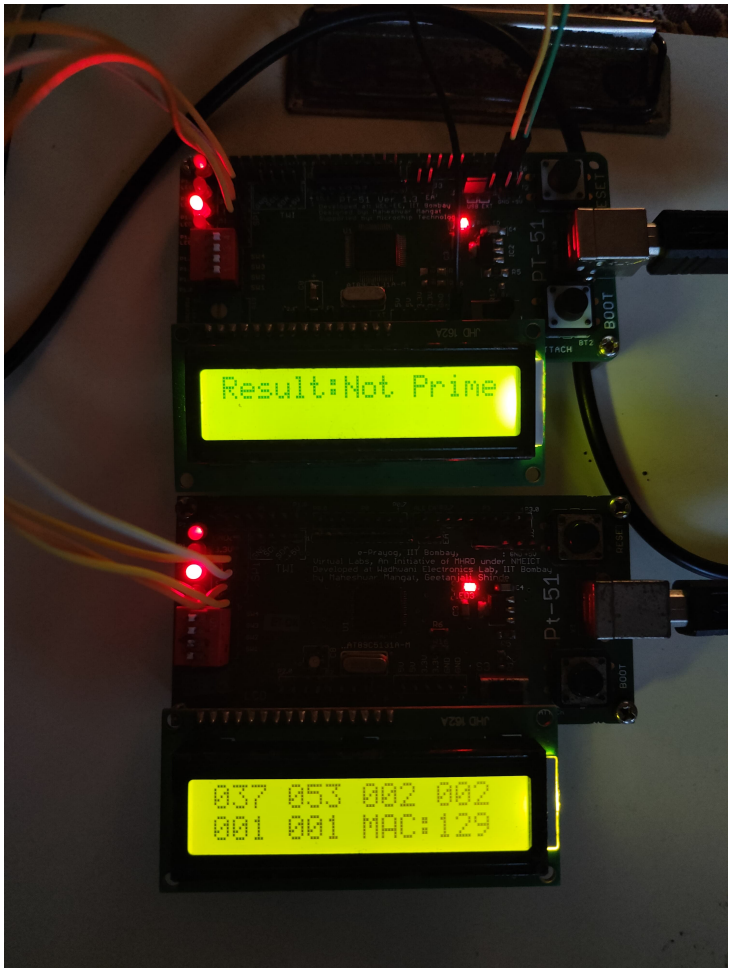
Result:Not Prime
037 053 002 002
001 001 MAC:129
PT-51
BOOT
Pt-51
BOOT

Figure 10

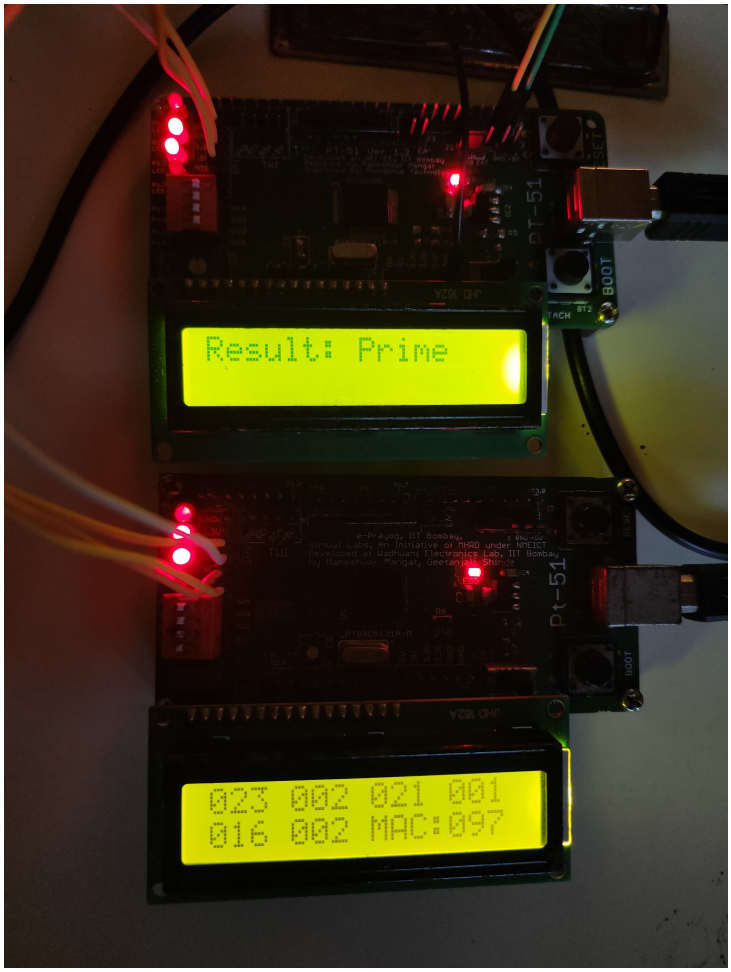

Figure 11