### Introduction

This project demonstrates how to establish serial communication between two PIC16F877A microcontrollers using the UART protocol. Specifically, Microcontroller A transmits a paragraph describing the MAX6952 display driver, while Microcontroller B receives the text, counts how many times the characters ‘D’, ‘E’, and ‘F’ appear, and then returns the original paragraph along with the counts. Finally, Microcontroller A uses a bit‐banged interface on PORTB (RB0, RB1, RB2) to send ASCII codes to the MAX6952, displaying the counts in the format D=x, E=y, F=z on a dot‐matrix LED.

Objectives of the assignment include,

1. Establishing a reliable UART communication between two PIC16F877A microcontrollers.
2. Transmit and receive ASCII characters.
3. Implement character counting in Microcontroller B for ‘D’, ‘E’, and ‘F’.
4. Display the counts on the MAX6952 driver from Microcontroller A.

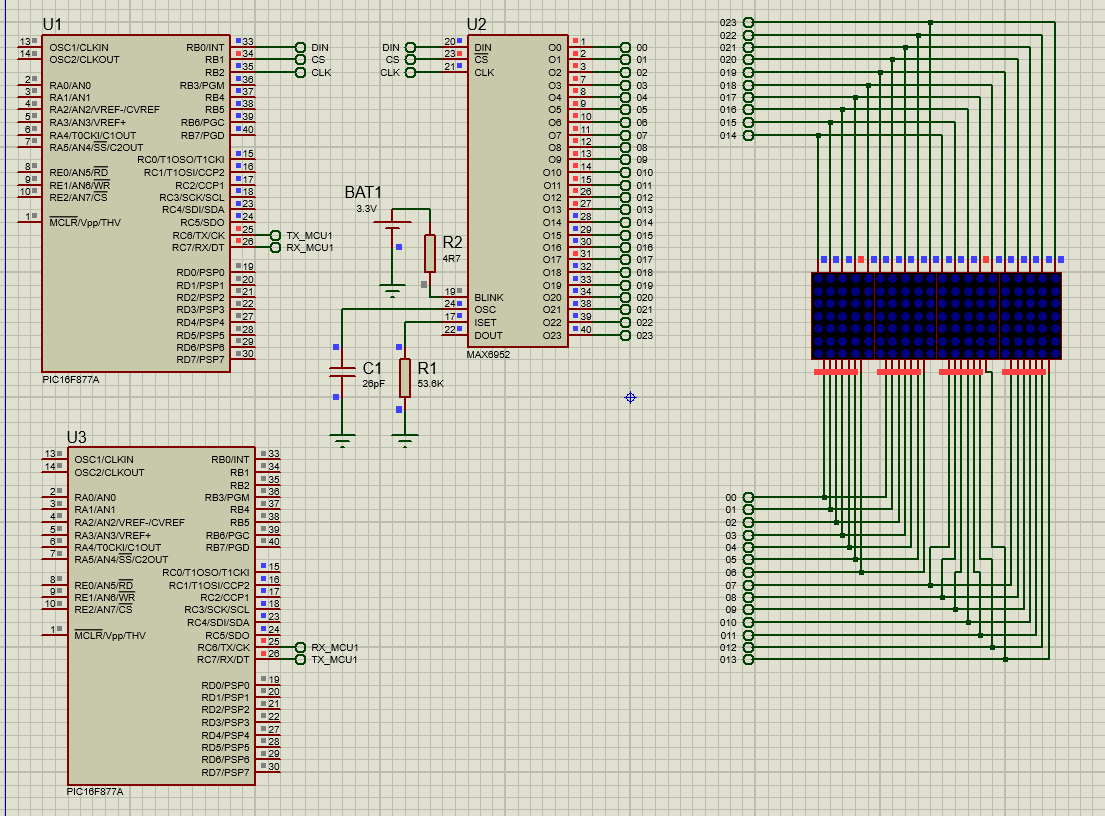
The following are required for the purpose of completing the assign,

1. MPLAB X IDE and XC8 Compiler for coding and building firmware for the PIC16F877A.
2. Proteus Simulation Tool for validation.

### Methodology

To enable UART communication, we consulted the PIC16F877A datasheet and connected RC6 (TX) on Microcontroller A to RC7 (RX) on Microcontroller B, and RC7 (RX) on Microcontroller A to RC6 (TX) on Microcontroller B. This cross-connection ensures that each MCU’s transmitter feeds into the other’s receiver. For the MAX6952 interface on Microcontroller A, we chose bit-banged signals on PORTB: RB0 goes to DIN, RB1 to CS, and RB2 to SCLK, allowing manual toggling of these lines in software. Both MCUs and the MAX6952 share a common ground for consistent signal references, and each device is powered by a +5 V supply with appropriate decoupling. Finally, for basic debugging and status indication, LED connections on PORTD were used to verify when characters such as ‘D’, ‘E’, or ‘F’ were detected by Microcontroller B. However it was later removed after the required functionality was achieved.

Below diagram shows that the components were connected to each other in proteus simulation software.



Proteus workspace

### Initializing the MAX6952 display driver

On initial power-up, all control registers are cleared, and the MAX6952 will be in shutdown mode. Prior to display use, programming is required. If not, it will be initialized to scan one digit, will not decode data in the data registers, and will have its intensity register set to its minimum value.

After initializing the preceding registers, the MAX6952 LED dot matrix display driver would be prepared to accept data for the 8-digit LED matrix display.

* The Shutdown Register must be configured for normal operation.
* The Decode-Mode Register must be configured for the required digits.
* The Intensity Register must be set to maximum LED intensity.
* The Scan-Limit Register must be configured to display 4 digits.
* The Display-Test Register must be configured for standard operation.

The register address map shows the addresses of the registers that need to be configured for the task.

*Initial powerup register status*

A table of numbers and symbols

Description automatically generated with medium confidence

X = Don’t care function

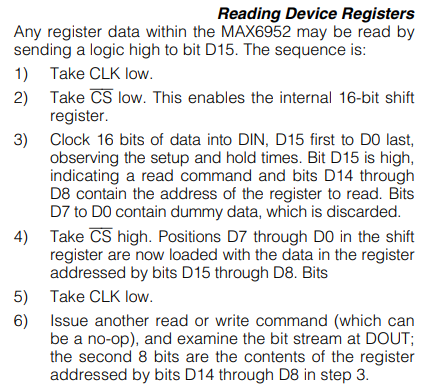
All don't care terms that were present in some data and address values were unilaterally set to the binary 0 value for versatility.

A diagram of a diagram of a hexagon

Description automatically generated

*Timing Diagram*

Each 16-bit address and data input must be entered according to the timing diagram provided by the manufacturer. Initially, the Chip Select (CS) must be set to logic high. Once it is set to logic low, the system becomes ready to receive all required data. After providing the Data Input (DIN) and Clock Input (CLK) signals, the Chip Select must again be set to logic high to complete the operation.

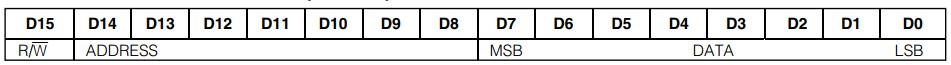


During the interval between the two logic high states of the Chip Select, each bit of data must be entered sequentially. For each bit, the Clock (CLK) is set to logic high, followed by a delay function to ensure proper timing. The Clock is then reset to logic low, and the delay function is invoked again. These steps are repeated until all bits of the 16-bit sequence are transferred, ensuring proper communication with the MAX6952.

### MAX6952 Display Driver

The MAX6952 contains 14 addressable digit and control registers, with the digit registers implemented as an on-chip 20x8 dual-port SRAM. These registers are directly addressable, allowing individual digits to be updated and retaining data as long as the supply voltage (V+) exceeds 2V (typically). The control registers include Decode Mode (to configure digit decoding), Intensity (to control LED brightness), Scan Limit (to set the number of scanned digits), Shutdown (to enable or disable the display), and Display Test (to activate a mode where all LEDs are turned on).

*Serial-Data Format (16 Bits)*



*Register Map*

A close-up of a register

Description automatically generated

### Configuration Register

The Configuration Register in the MAX6952 controls its operational modes, such as enabling or disabling the display through Shutdown Mode, allowing blinking mode for alarm or notification purposes, and configuring the number of active digits being scanned through Scan Mode Configuration. The register retains its settings as long as the supply voltage (V+) is above the minimum operating voltage. Programming the Configuration Register is a crucial step in initializing and customizing the display's behavior.

A close-up of a register

Description automatically generated

The S bit in the configuration register selects shutdown or normal operation (Table 7). In shutdown mode, the scan oscillator ceases operation, and all LED segment current sources are pulled to ground. This mode can be used for power-saving purposes or to implement a display blinking alarm. The display driver remains programmable even while in the shutdown state.

A white rectangular box with black letters and numbers

Description automatically generated with medium confidence

Therefore, the configuration register bit 0 need to be set as 0.

### Display Intensity Register

The display brightness in the MAX6952 is controlled by a 10kΩ resistor connected externally between the V+ and ISET pins. Additionally, the MAX6952 permits digital control of the display intensity via uploaded code. This digital control is provided by an internal pulse width modulator (PWM), which regulates the display's brightness in 32 levels, ranging from 1/32 of the maximum intensity to full brightness.

*Intensity Register Format*

A table with numbers and letters

Description automatically generated

A black and white document with numbers

Description automatically generated

To ensure the intensity is maximum, the register value can be set to 0xFF.

### Display Test Register

The Display Test Mode in the MAX6952 enables all LEDs by overriding normal operation. A logic high input activates the Display Test Mode, turning on all LEDs, while a logic low input restores the normal operation mode. Once configured, the MAX6952 remains in Display Test Mode until it is explicitly switched back to normal operation.

*Display-Test Register Format*

A white rectangular box with black text

Description automatically generated

### Next the display test register value is set to 0x00 to ensure normal operation

### Dot Matrix Display

A dot matrix display is used to display the count number. A dot matrix display consists of a grid of LED dots arranged in rows and columns. By selectively lighting up specific combinations of LEDs, it can form numbers, characters, and even simple graphics. Dot matrix displays can either be Common Cathode (CC) or Common Anode (CA).

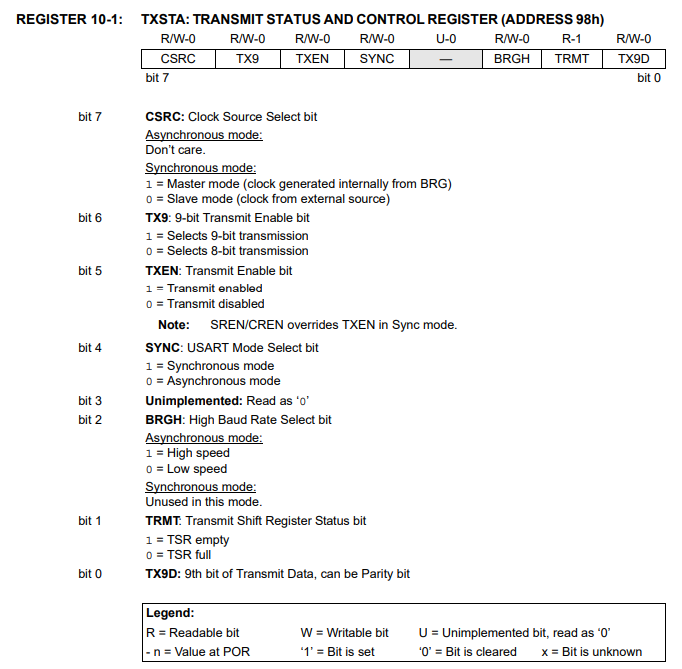
In a common cathode dot matrix display, all the cathode (K) connections of the LEDs in each row are tied together and connected to ground or zero-volts. In a common anode dot matrix display, all the anode (A) connections of the LEDs in each column are joined together to a positive voltage supply.

A diagram of a diagram of a line of triangles and arrows

Description automatically generated with medium confidenceA diagram of a diagram of electricity

Description automatically generated

### UART Communication



A black and white text on a white background

Description automatically generated

A screenshot of a computer program

Description automatically generated

The above 2 registers need to be configured to,

• Enable 8 bit transmission (TX9=0)

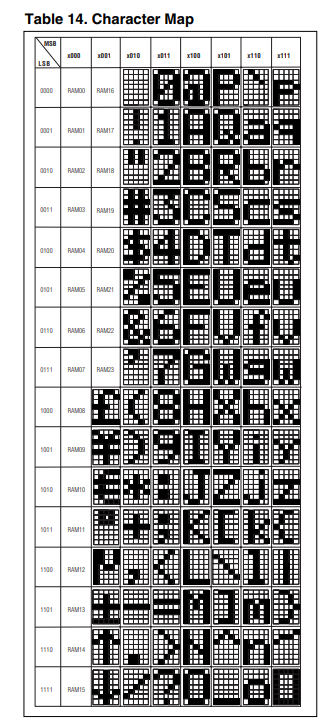
• Asynchronous communication (SYNC=0)

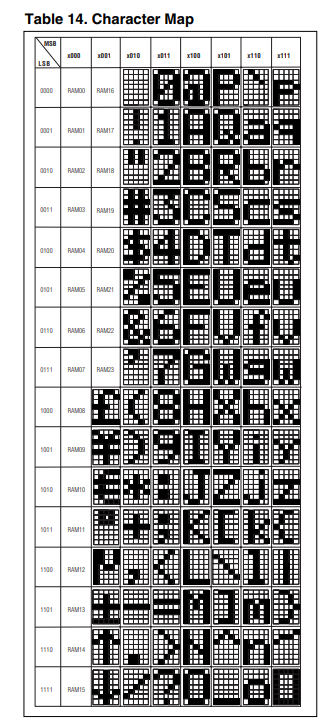
• Enable transmittion (TXEN=1)

• Enable serial port (SPEN=1)

• Continuous receive (CREN=1)

### Character Mapping





Accordingly, required hex values were fed to the MAX6952 IC for displaying certain characters

|  |  |
| --- | --- |
| Character | Hex value |
| D | 0x44 |
| E | 0x45 |
| F | 0x46 |
| 0 | 0x30 |
| 1 | 0x31 |
| 2 | 0x32 |
| 3 | 0x33 |
| 4 | 0x34 |
| 5 | 0x35 |
| 6 | 0x36 |
| 7 | 0x37 |
| 8 | 0x38 |
| 9 | 0x39 |
| = | 0x3D |
| , | 0x2C |
| . | 0x2E |

**Microcontroller A**

The 1st microcontroller that sends a string of text and receives count and displays the count on the LED Matrix through the MAX6952 IC

#include <xc.h>

#pragma config FOSC  = HS

#pragma config WDTE  = OFF

#pragma config PWRTE = OFF

#pragma config BOREN = OFF

#pragma config LVP   = OFF

#pragma config CPD   = OFF

#pragma config WRT   = OFF

#pragma config CP    = OFF

#define \_XTAL\_FREQ 20000000

// ------------ Prototypes ------------

*void* UART\_Init(*long* *baudRate*);

*void* UART\_TxChar(*char* *ch*);

*char* UART\_RxChar(*void*);

*void* send16Bit(*char* *address*, *char* *dataByte*);

*void* displayMessage(const *char* \**msg*);

*void* displayCount(*char* *letter*, *char* *count*, *char* *suffix*);

*void* UART\_Init(*long* *baudRate*)

{

    // RC6 = TX (output), RC7 = RX (input)

    TRISC = 0x80;

    PORTC= 0x00;

    TX9=0;

    SYNC=0;

    TXEN=1;

    SPEN=1;

    CREN=1;

    SPBRG = (*int*)((20000000 / (64 \* (*baudRate* + 1))));

}

*void* UART\_TxChar(*char* *ch*)

{

    while(TXIF==0);

    TXIF = 0;

    TXREG = *ch*;

}

*char* UART\_RxChar(*void*)

{

    while(RCIF==0);

    RCIF = 0;

    return RCREG;

}

// We assume RB0 -> DATA, RB1 -> CS, RB2 -> CLK

*void* send16Bit(*char* *address*, *char* *dataByte*)

{

*int* word = ((*int*)*address* << 8) | (*dataByte* & 0xFF);

*int* i;

    // Bring CS high, then low

    RB1 = 1;

    \_\_delay\_ms(50);

    RB1 = 0;

    \_\_delay\_ms(50);

    for(i = 0; i < 16; i++)

    {

        if(word & 0x8000)

        {

            RB0 = 1;

        }

        else

        {

            RB0 = 0;

        }

        \_\_delay\_ms(50);

        // Clock high

        RB2 = 1;

        \_\_delay\_ms(50);

        // Clock low

        RB2 = 0;

        \_\_delay\_ms(50);

        word <<= 1;

    }

    RB1 = 1;

    \_\_delay\_ms(50);

}

*void* display4Chars(*char* *c1*, *char* *c2*, *char* *c3*, *char* *c4*)

{

    send16Bit(0x20, *c1*);

    send16Bit(0x21, *c2*);

    send16Bit(0x22, *c3*);

    send16Bit(0x23, *c4*);

}

*void* displayCount(*char* *letter*, *char* *count*, *char* *suffix*)

{

    // letter, '=', digit, suffix

    display4Chars(*letter*, 0x3D, (0x30 + *count*), *suffix*);

    // Clear  display

    display4Chars(0x20, 0x20, 0x20, 0x20);

}

*void* main(*void*)

{

    // Configure PORTB for bit output

    TRISB = 0x00;

    PORTB = 0x00;

    send16Bit(0x04, 0x81);  // Configuration

    send16Bit(0x01, 0xFF);  // Intensity

    send16Bit(0x02, 0xFF);  // Intensity 32

    send16Bit(0x07, 0x00);  // Display test off

    UART\_Init(9600);

    const *char* text[] =

    "The MAX6952 is a compact serial input/output display driver designed to interface "

    "microcontrollers with dot-matrix LED displays. This chip supports SPI and various other serial "

    "communication protocols, requiring only three control lines from your microcontroller to "

    "effectively drive the display.\n";

*int* i = 0;

    while (text[i] != '\0')

    {

        UART\_TxChar(text[i]);

        UART\_RxChar();

        i++;

    }

*char* c;

*char* input[2];

    while(1)

    {

        c = UART\_RxChar();

        if(c=='#'){

            for(i=0; i<3;i++){ // received DEF

                c = UART\_RxChar();

                input[i]=c;

            }

            // Display "D=x,"

            displayCount(0x44, input[0], 0x2C); // ','=0x2C

            // Display "E=x,"

            displayCount(0x45, input[1], 0x2C); // ','=0x2C

            // Display "F=x."

            displayCount(0x46, input[2], 0x2E); // '.'=0x2E

        }

    }

}

In this code, the PIC16F877A initializes its UART (via UART\_Init) on pins RC6 (TX) and RC7 (RX) and then transmits a long paragraph stored in const char text[]. Each character is sent out one by one using UART\_TxChar, with a small wait in the loop by receiving a character from the other MCU (to synchronize or simply acknowledge each sent character).

Simultaneously, the program sets up the MAX6952 driver through a bit-banged routine (send16Bit) on the pins RB0 (DATA), RB1 (CS), and RB2 (CLK). The code first writes configuration and intensity registers (e.g., addresses 0x04, 0x01, 0x02, 0x07) to initialize the display.

Once the text transmission is complete, the microcontroller listens for a specific marker ('#') from the second microcontroller, signaling that three bytes of counts (for ‘D’, ‘E’, and ‘F’) will follow. These counts are stored in an array (input[3]), and the function displayCount is used to show each result on the MAX6952 by writing ASCII codes for “D=x,”, “E=y,” and “F=z.” to consecutive column addresses. Each displayed sequence is briefly cleared by sending space characters (0x20) to the same addresses, giving a neat update on the LED display.

**Microcontroller B**

The 2nd microcontroller that receives the string of text, retransmit the charcters along with the counts of ‘D’,’E’ and ‘F’

#include <xc.h>

#pragma config FOSC  = HS

#pragma config WDTE  = OFF

#pragma config PWRTE = OFF

#pragma config BOREN = OFF

#pragma config LVP   = OFF

#pragma config CPD   = OFF

#pragma config WRT   = OFF

#pragma config CP    = OFF

#define \_XTAL\_FREQ 20000000

// -------------- Prototypes --------------

*void* UART\_Init(*long* *baudRate*);

*void* UART\_TxChar(*char* *ch*);

*char* UART\_RxChar(*void*);

*void* UART\_Init(*long* *baudRate*)

{

    // RC6 = TX (output), RC7 = RX (input)

    TRISC = 0x80;

    PORTC= 0x00;

    TX9=0;

    SYNC=0;

    TXEN=1;

    SPEN=1;

    CREN=1;

    // Baud rate: SPBRG = (Fosc / (64 \* baud)) - 1  [for BRGH=0]

    SPBRG = (*int*)((20000000 / (64 \* (*baudRate* + 1))));

}

*void* UART\_TxChar(*char* *ch*)

{

    while(TXIF==0);

    TXIF = 0;

    TXREG = *ch*;

}

*char* UART\_RxChar(*void*)

{

    while(RCIF==0);

    RCIF = 0;

    return RCREG;

}

*void* main(*void*)

{

*char* input[10];

*char* c;

*int* i=0;

*int* countD=0, countE=0, countF=0;

    UART\_Init(9600);

    while(1)

    {

*char* c = UART\_RxChar(); // receiving from MCU1

        UART\_TxChar(c); // transmit back

        if(c == '\n')

        {

            break;  // Stop reading

        }

        if(c == 'D'){

            countD++;

        }

        else if(c == 'E'){

            countE++;

        }

        else if(c == 'F'){

            countF++;

        }

    }

    UART\_TxChar('#'); // to keep track of the counts

    UART\_TxChar(countD);

    UART\_TxChar(countE);

    UART\_TxChar(countF);

}

This code configures a second PIC16F877A for UART reception. It also runs at 9600 baud and continuously reads incoming characters from Microcontroller A. Every character received is immediately echoed back (UART\_TxChar(c)) so that the sender can verify successful transmission. The code monitors each character to see if it matches 'D', 'E', or 'F', and increments local counters (countD, countE, and countF), respectively, whenever one of those letters is encountered. The loop terminates when it detects a newline '\n', marking the end of the paragraph. Afterwards, the code transmits a '#' character to inform the first microcontroller that the final counts are about to be sent. It then sends out the three counts in raw form (i.e., the integer values for the total occurrences of ‘D’, ‘E’, and ‘F’). By doing so, the second microcontroller effectively provides both an echo of the original text and a summary of how many times these specific characters appeared.

### Results

Upon running the system, Microcontroller A successfully transmitted the specified paragraph regarding the MAX6952 display driver. Microcontroller B correctly received and echoed each character, incrementing internal counters whenever it encountered the letters ‘D’, ‘E’, or ‘F.’ Once the paragraph was complete (signaled by a newline), Microcontroller B sent back a marker ('#') followed by the three byte values indicating the total count of each letter. On Microcontroller A’s side, these counts were correctly interpreted and displayed on the LED matrix—managed by the MAX6952—as “D=x,” “E=y,” and “F=z.” For instance, if the paragraph contained five occurrences of ‘D,’ two of ‘E,’ and two of ‘F,’ the display showed “D=5,” “E=2,” and “F=2.” This sequence verified that both the UART link and the bit‐banged interface to the MAX6952 were operating correctly.

Transmitted string: “The MAX6952 is a compact serial input/output display driver designed to interface microcontrollers with dot-matrix LED displays. This chip supports SPI and various other serial communication protocols, requiring only three control lines from your microcontroller to effectively drive the display.”

Results of the transmitted string which has ‘D’, 1 ’E’ and no ’F’

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer game

Description automatically generated

### Conclusion

In conclusion, this project successfully demonstrated the transmission and processing of data between two PIC16F877A microcontrollers via UART. Microcontroller A sent a detailed paragraph on the MAX6952 display driver and later displayed the counts of specific characters (‘D’, ‘E’, and ‘F’) on the MAX6952 using a bit-banged interface. Microcontroller B reliably echoed the incoming text and implemented the logic necessary to count the target characters, then returned those counts for display. Through this work, we have validated both the serial communication link and the integration of an external display driver, showcasing a practical embedded application that combines hardware interfacing, software design, and the handling of textual data within limited resources.

### References

1. Microchip Technology Inc., *PIC16F877A Data Sheet*, 2001. [Online]. Available:  
   https://www.microchip.com/en-us/product/PIC16F877A
2. Microchip Technology Inc., *MPLAB® XC8 C Compiler User’s Guide*, 2018. [Online]. Available:  
   <https://www.microchip.com/en-us/development-tools-tools-and-software/mplab-xc-compilers>
3. Analog Devices Inc. (Maxim Integrated), *MAX6952: Compact Serial Input/Output Display Driver*, [Online]. Available: https://www.analog.com/en/products/max6952.html