Serial Terminal EEPROM Interface

Matthew Beckler
beck0778@umn.edu
EE2361 Lab 007

April 25, 2006

Abstract

The PIC18F452 has a built-in 256-byte data EEPROM, which is accessible through a special file register (SFR). In this experiment, we have created a user interface, using the PC's serial port with the Hyperterminal software package, and the PIC's EEPROM. There are a number of commands the user can enter at the computer, including commands to display the current EEPROM contents, enter data, or erase the EEPROM entirely.

Introduction

The software for the lab experiment has a fairly simple main control loop. After initializing the serial USART and EEPROM interfaces, the program enters into an input-based while loop. Each time through the input-loop, the processor prints the prompt (`->`), and waits for input from the user. Based on the user's input, the processor performs one of the following actions:

1. **D** - Dump EEPROM contents in Hexadecimal format.
2. **T** - Print EEPROM contents as a string.
3. **E** - Erase EEPROM.
4. **S** - Enter data in string format.
5. **H** - Enter data in hexadecimal format.

After completing the directed task, the processor returns the user to the prompt, waiting for the next command.

Observations

In this section, I will be describing the processor's detailed operation for each of the five commands. First, is a description of the general processor setup, as well as a description of all the helper functions.

A note about hexadecimal versus ASCII encoding is needed at this time. In the standard BCD method of storing values in memory, a nibble (4-bits) can hold one hexadecimal digit, of 0-9 or A-F. When transmitting characters

through the serial USART, the ASCII encoding scheme is used. In ASCII, the digits 0-9 are represented by the values 0x30 - 0x39, the uppercase letters A-F are 0x41 - 0x5B, and the lowercase letters a-f are 0x61 - 0x7B. This difference will play an important role later in this experiment.

The serial USART is setup in much the same way as the previous experiment. It has been set for 9600 baud, in 8-bit transmit and receive. The two accessor functions, `putch()` and `getch()`, have returned as well. To facilitate the transmission of data in human-readable hexadecimal format, the function `nibbleToHex()` has been written, which translates between the ASCII representation of a hexadecimal digit and the number's true value. This function converts a nibble, which will have a value between 0 and 15, into it's equivalent ASCII character (0-9, A-F). This allows for a very easy transmission of the EEPROM's data in hexadecimal format. For the hexadecimal input, a function, named `validChar()`, has been created that determines if the specified character is a valid hexadecimal digit, namely 0-9 or A-F. It has been designed to accommodate both uppercase and lowercase letters. To actually convert these characters into their proper values (0-15), the function `hexToNibble()` has been created. It translates the ASCII-encoded hexadecimal digits into their actual numeric values.

At startup, the data EEPROM is also configured. It is setup for both reading and writing, and is initially erased to all zeros. The procedure for reading values from the EEPROM is fairly straightforward, and is detailed later in this document. Reading data from the EEPROM happens immediately, as there is no delay between setting the 'READ' bit and reading the data out of the register.

1. Set the address of the value to read: `EEADR`.
2. Set the read-bit: `EECON1bits.RD = 1`.
3. Read the value out of the data register: `EEDATA`.

Writing to the EEPROM is more involved, and also takes more time. The time required for a successful write to the EEPROM depends on supply voltage and temperature, and will also vary between processors. The write cycle is controlled by an on-chip timer, which is out of the user's control. Before attempting the write cycle, it is recommended that the user disable interrupts, however this application does not use interrupts, they have not been changed. First, the registers `EEADR` and `EEDATA` must be loaded with the proper values, namely the desired address of the write, and the desired data to write. Then, in a very strange and picky sequence, you must write the values `0x55` and `0xAA` into the register `EECON2`. This appears to be completely superficial and pointless, but of course, I have forgotten that everything the Microchip engineers do is always perfectly correct, so I apologize for my insolence. Only after writing the values into `EECON2`, may you finally enable the write bit, `EECON1bits.WR`. As stated before, the write cycle takes a variable amount of time, which is going to be longer than 100ns. To ensure we do not start writing a new value until the EEPROM is finished writing the previous value, we have introduced a delay loop which waits until the `PIR2bits.EEIF` returns to 0, which signifies the end of the write cycle.

Dump Data in Hexadecimal Format

When the user enters the character 'D' or 'd', the processor dumps the EEPROM's data to the terminal in hexadecimal format. To accomplish this, the processor iterates through the rows and columns of the EEPROM's address space, 16 addresses per row, leaving 16 columns total. For each memory address, the processor converts the two nibbles of that particular byte into their respective ASCII representations, which are then transmitted to the computer. Two spaces are printed between each byte, with a carriage return and newline printed after each row.

Print EEPROM contents as a string

When the user enters the character 'T' or 't', the processor prints the EEPROM's contents not as hexadecimal digits, but it assumes that raw ASCII was stored into the EEPROM. It transmits the raw EEPROM data with no translation to the computer, only stopping when a null character (0x00) is reached.

Erase EEPROM

To erase the EEPROM, the user enters the character 'E' or 'e'. When it receives one of these characters, the processor starts a for loop through the entire EEPROM address space, writing the value 0x00 to each byte. It then transmits the message `EEPROM Erased` to the computer, to indicate that the EEPROM has been successfully erased.

Enter data in string format

To enter data into the processor, the user has the option of either hexadecimal input, or ASCII character input. If one of the characters 'S' or 's' is entered, the processor starts waiting for consecutive digits to be transmitted by the computer, waiting for a carriage return to signal the end of input. After receiving the final carriage return, the processor writes the value 0x00 after the end of the user's data, signalling the end of string in memory.

Enter data in hexadecimal format

When the user enters the character 'H' or 'h', the processor starts waiting for a pair of valid hexadecimal digits. If either of these transmitted digits is invalid, the processor returns to the prompt. If both digits are valid, the processor writes them into the next EEPROM address.

Conclusion

Altogether, this project has been a success. The EEPROM interface works quite well, and the serial terminal interface provides a familiar way for humans to interact with a low-level device. The five commands provide a useful set of actions to interact with the EEPROM, both in human-readable ASCII strings, as well as the lower-level hexadecimal numbers.

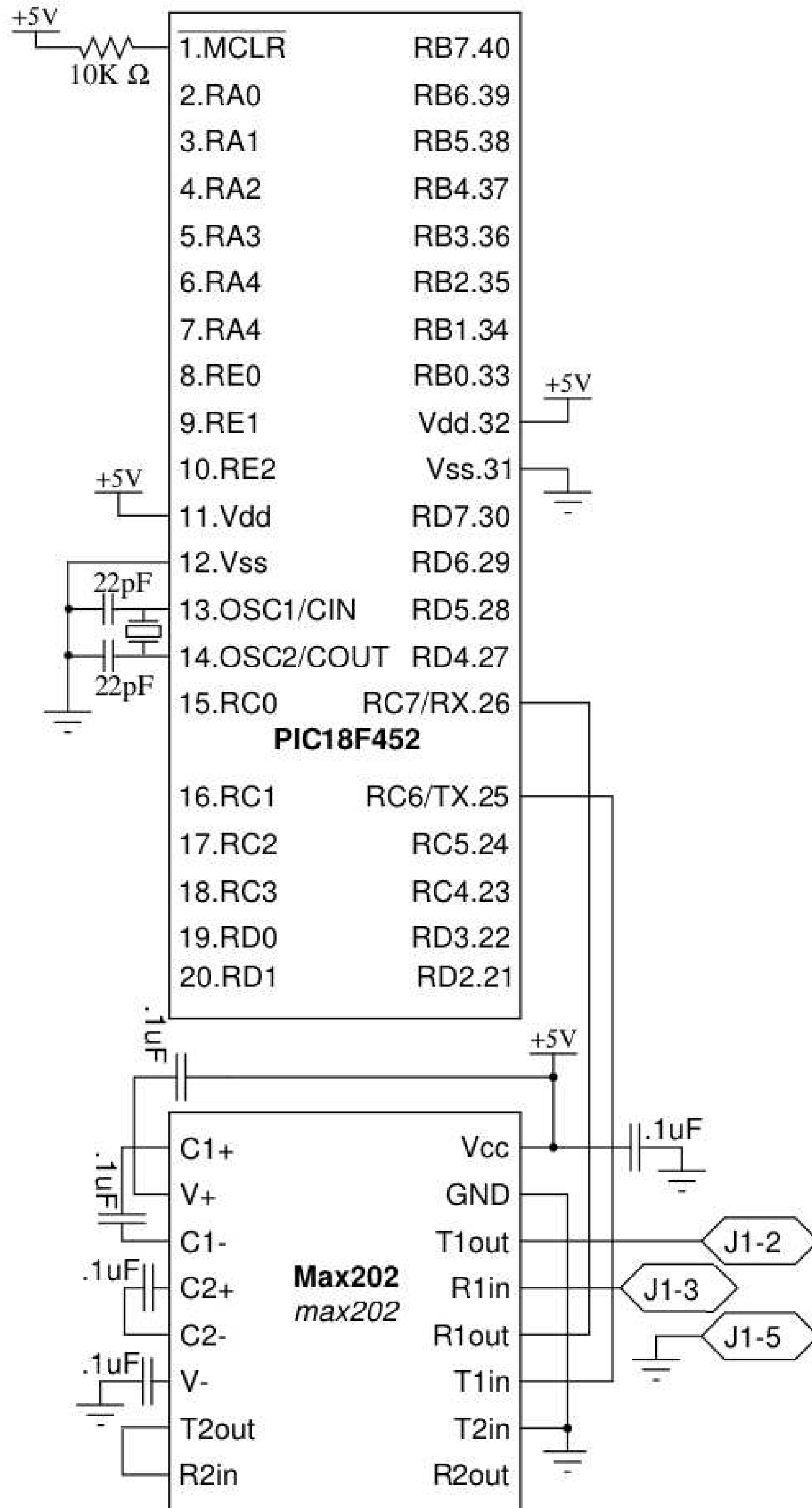

Figure 1: Circuit Schematic - Courtesy Professor Rennolet

C Language Program

```
#include <p18f452.h>
#pragma config OSC=HSPLL, WDT=OFF, BOR=OFF, PWRT=ON

void putch(char byte)
{
    while (!PIR1bits.TXIF);
    TXREG = byte;
}

char getch(void)
{
    while (!PIR1bits.RCIF);
    return RCREG;
}

char nibbleToHex(char x)
{
    x += 0x30;
    if (x > 0x39)
    {
        x += 7;
    }
    return x;
}

char hexToNibble(char x)
{
    if (x >= 0x30 && x <= 0x39)
    { //regular numeric digit
        return (x - 0x30);
    }
    else if (x >= 0x41 && x <= 0x5B)
    { //uppercase hex
        return (x - 55);
    }
    else if (x >= 0x61 && x <= 0x7B)
    { //lowecase hex
        return (x - 87);
    }
}

char validChar(char x)
{
    //checks if char x is a valid hex digit
    //0 = 0x30
    //9 = 0x39
    //A = 0x41
    //F = 0x46
```

```
    //a = 0x61
    //f = 0x66

    if (x >= 0x30 && x <= 0x39)
    { //digit
        return 1;
    }
    else if (x >= 0x41 && x <= 0x46)
    { //uppercase hex
        return 1;
    }
    else if (x >= 0x61 && x <= 0x66)
    { //lowercase hex
        return 1;
    }
    else
    {
        return 0;
    }
}

void main(void)
{
    //char x = hexToNibble(0x34);

    char welcomeMessage[] = "EEPROM Interface";
    char erasedMessage[] = "EEPROM Erased";
    int i, rows, cols;
    char input, validInput, temp;

    //initialize serial registers
    TXSTAbits.TXEN = 1;
    RCSTAbits.SPEN = 1;
    RCSTAbits.CREN = 1;
    SPBRG = 0x40;

    //initialize eeprom interface
    EECON1bits.EEPGD = 0;
    EECON1bits.CFGS = 0;
    EECON1bits.WREN = 1;
    for (rows = 0; rows < 256; rows++)
    {
        EEADR = rows;
        EEDATA = 0;
        EECON2 = 0x55;
        EECON2 = 0xAA;
        EECON1bits.WR = 1;
        while (!PIR2bits.EEIF);
        PIR2bits.EEIF = 0;
    }
```

```
//display welcome message
for (i = 0; welcomeMessage[i] != '\0'; i++)
{
    putch(welcomeMessage[i]);
}
putch(0x0D);
putch(0x0A);

while(1)
{
    //write prompt
    putch('-');
    putch('>');
    putch(' ');
    input = getch();
    putch(input); //echo character
    if (input == 0x0D)
    {
        putch(0x0A);
    }
    putch(0x0D);
    putch(0x0A);

    switch (input)
    {
        case 'd':
        case 'D':
            //Display hexadecimal output
            EEADR = 0;
            for (rows = 0; rows < 16; rows++)
            {
                for (cols = 0; cols < 16; cols++)
                {
                    EEADR = rows*16 + cols;
                    EECON1bits.RD = 1;

                    putch(nibbleToHex(EEDATA >> 4));
                    putch(nibbleToHex(EEDATA & 0xF));

                    putch(' ');
                    putch(' ');
                }
                putch(0x0D);
                putch(0x0A);
            }
            break;
        case 't':
        case 'T':
            //Display sTring output
```

```
        EEADR = 0;
        for (rows = 0; rows < 256; rows++)
        {
            EEADR = rows;
            EECON1bits.RD = 1;
            if (EEDATA != 0)
            {
                putch(EEDATA);
            }
            else
            {
                break;
            }
        }
        putch(0x0D);
        putch(0x0A);
        break;
    case 'e':
    case 'E':
        //Erase EEPROM
        for (rows = 0; rows < 256; rows++)
        {
            EEADR = rows;
            EEDATA = 0;
            EECON2 = 0x55;
            EECON2 = 0xAA;
            EECON1bits.WR = 1;
            while (!PIR2bits.EEIF);
            PIR2bits.EEIF = 0;
        }

        for (i = 0; erasedMessage[i] != '\0'; i++)
        {
            putch(erasedMessage[i]);
        }
        putch(0x0D);
        putch(0x0A);
        break;
    case 's':
    case 'S':
        //enter String input
        EEADR = 0;
        while ((input = getch()) != 0x0D)
        {
            putch(input);
            EEDATA = input;
            EECON2 = 0x55;
            EECON2 = 0xAA;
            EECON1bits.WR = 1;
            while (!PIR2bits.EEIF);
```

```
            PIR2bits.EEIF = 0;
            EEADR++;
        }

        EEDATA = 0;
        EECON2 = 0x55;
        EECON2 = 0xAA;
        EECON1bits.WR = 1;
        while (!PIR2bits.EEIF);
        PIR2bits.EEIF = 0;

        putch(0x0D);
        putch(0x0A);
        break;
    case 'h':
    case 'H':
        //enter Hexadecimal input
        EEADR = 0;
        validInput = 1;
        temp = 0;
        input = 0;
        while (validInput == 1)
        {
            putch(0x0D);
            putch(0x0A);
            putch('=');
            putch(' ');
            input = getch();
            putch(input);
            if (validChar(input) == 0)
            {
                validInput = 0;
            }
            else
            {
                //valid character
                temp = input; //save first digit
                input = getch();
                putch(input);
                if (validChar(input) == 0)
                {
                    validInput = 0;
                }
                else
                {
                    EEDATA = (hexToNibble(temp) << 4) + hexToNibble(input);
                    EECON2 = 0x55;
                    EECON2 = 0xAA;
                    EECON1bits.WR = 1;
                    while (!PIR2bits.EEIF);
```

```
                        PIR2bits.EEIF = 0;
                        EEADR++;

                        while (getch() != 0x0D);
                    }
                }
            }

            putch(0x0D);
            putch(0x0A);
            break;
        }
    }
}
```