Integrating Hardware and Software Filtering in Embedded System Audio Data Processing: An Embedded Systems Course Project

Vincent Winstead
Minnesota State University, Mankato

Abstract

This paper describes a course laboratory project for an embedded systems course. The project is intended to provide a real world embedded development task for the students to accomplish in a few week time using a predefined microcontroller and suggested circuit components. The task combines audio sound recording, off-processor storage and filtered audio data replay. The paper includes a brief summary of the course concepts and the particular topics related to the project, an overview of the project goals and suggested circuit and constraints and a possible solution to satisfy the task goals including a complete circuit design and software design. Finally, the paper concludes with a discussion of the in-class results of assigning this project to students, their feedback and possible future changes to enhance the learning experience in future offerings.

Project Summary

The project was designed around a target processor (microcontroller) from Microchip Co. to exercise the student's knowledge of the SPI (Serial Peripheral Interface) protocol and its implementation in the processor. In addition, the students were required to apply knowledge of digital and analog circuit design to complete a working demonstration. The embedded systems course gives the students an opportunity to put to practice the skills gained through previous coursework and electronics laboratory experimentation. This paper describes the course concepts associated with the audio system project, the project itself including the instructor's example hardware design intended as a possible interfacing option for the students, and some conclusions based on multiple course offerings of this project.

The embedded systems course covers many topics concerning the interfacing of sensors, actuators, peripherals, I/O (Input/Output) and communication methods within an embedded design. The course is conducted using a combination of lecture material covering embedded hardware and software theory as well as project based laboratory experiences. This provides an environment for project teams to apply the concepts from lecture taking design specifications from prototype design to implementation, test, and

demonstration. All projects have a required demonstration to showcase the results from team activities.

The audio project discussed in this paper is described below. All students are provided with the following project specifications:

Project Description

This project involves the construction of an external memory circuit using a low voltage Flash memory device. The memory interface is implemented using the Serial Peripheral Interface (SPI) protocol. A circuit diagram is included. The goal of the project is to enable data recording and play back for digital and/or analog data at frequencies up to 3 kHz. Each team has two options for the completion of the project, one of which must be completed. All graduate students must complete Option 2.

Option 1: Record variable frequency logic data (50% duty cycle) for five seconds with signal frequencies up to 3 kHz. Then, play the recorded data through the protoboard speaker using appropriate interface circuitry. An eight Ohm speaker requires a driver and cannot be adequately driven directly from the pic18f8680 processor. The record and playback interface must occur using the USART peripheral and HyperTerminal. All circuits must be powered using the SSE 8680 5V source.

Option 2: Record analog voice data (Note: The diagram for a possible microphone interface circuit is available) for five seconds. Then, play the recorded data through the protoboard speaker using sufficient filtering and interface circuitry. An eight Ohm speaker requires a driver and cannot be adequately driven directly from the pic18f8680 processor. The record and playback interface must occur using the USART peripheral and HyperTerminal. All circuits must be powered using the SSE 8680 5V source.

These specifications do not come with many restrictions and this is to allow the student project teams to add an element of design to the project completion. For example, the project discussed in this paper explicitly requires consideration the SPI protocol and its implementation in the target processor, but also required the students to research (on their own) viable options for efficiently storing digitized data with a limited bandwidth and converting the stored data back to sound. In particular, Option 2 requires the storage and playback of voice data. This requires the students to research the appropriate sampling frequency, data storage event frequency, data filtering and playback. The students were not required to complete the data conversion (digital to analog) for the playback using a hardware converter (Note: The target PIC microcontroller does not have a built in D/A hardware converter.) but were free to determine and implement possible hardware and /or software solutions. The instructor example solution will describe a software conversion solution.

The target processor is one of the class of PIC18 8-bit (data path) microcontrollers available through Microchip Co. The engineering and technology department has

utilized PIC microcontrollers for multiple courses and laboratory experiences as well as encouraged their use in senior design experiences due to their low relative cost, freely available student versions of a fully integrated IDE (Integrated Development Environment) and because of the department's investment in programming hardware used to write/erase the onboard Flash memory. The embedded course covers an overview and tutorial of the Microchip IDE software package called MPLAB. The IDE allows development in Assembly Language as well as C-programming. The course is taught emphasizing programming in C and the solution code described in the paper is written in C. The project requires interfacing with a SPI Flash external memory device (M25P16). The device provided is from STMicroelectronics however, compatible devices can also be used with possibly small changes to the software. This device was specifically chosen for four reasons: 1) It is an SPI interfaced memory device requiring the implementer to carefully read the specifics of the datasheet, 2) It can be operated at speeds which exceed the instruction rate of the target microcontroller so that the target processor is the speed limiter for the project, 3) It is a 3.3V device requiring care to interface with the target processor which is a 5V device, 4) It is a surface mount device (SOIC) requiring the student teams to exercise surface mount soldering. To power a 3.3V device, one can use a 3.3V voltage regulator. The LT1121 was chosen based on the current sourcing capability, robustness and the low dropout capability. As with the SPI memory device, this device was chosen with the hope that the limitations associated with the project designs/prototyping are due to those aspects of the project controlled by the students.

The diagram below was provided to the students along with the project description. The top circuit is used only with the Option 2 portion of the project and it is a typical combined level shifting signal gain design found in the literature and within many electronics suppliers application documentation. This design was influenced by the WM Series Electret Circuit available on the www.Digikey.com website. The bottom circuit was influenced by the information in the datasheets for the memory and regulator devices. Resistors are used between all signals coming from the target processor to the SPI memory device to drop the excess voltage from the target processor. This is not the only way to provide an interface and in fact is not very robust since the current draw of the SPI memory device varies between different manufacturers and the resistor values need to be configured specifically for the specific device. There are devices available which specifically facilitate this interface such as 74LVC4245 Octal Bus Transceiver and level shifter. Alternatively, one could use opto-isolators to interface 5V to 3.3V I/O. Other options are also possible and the students were encouraged to attempt other implementations.

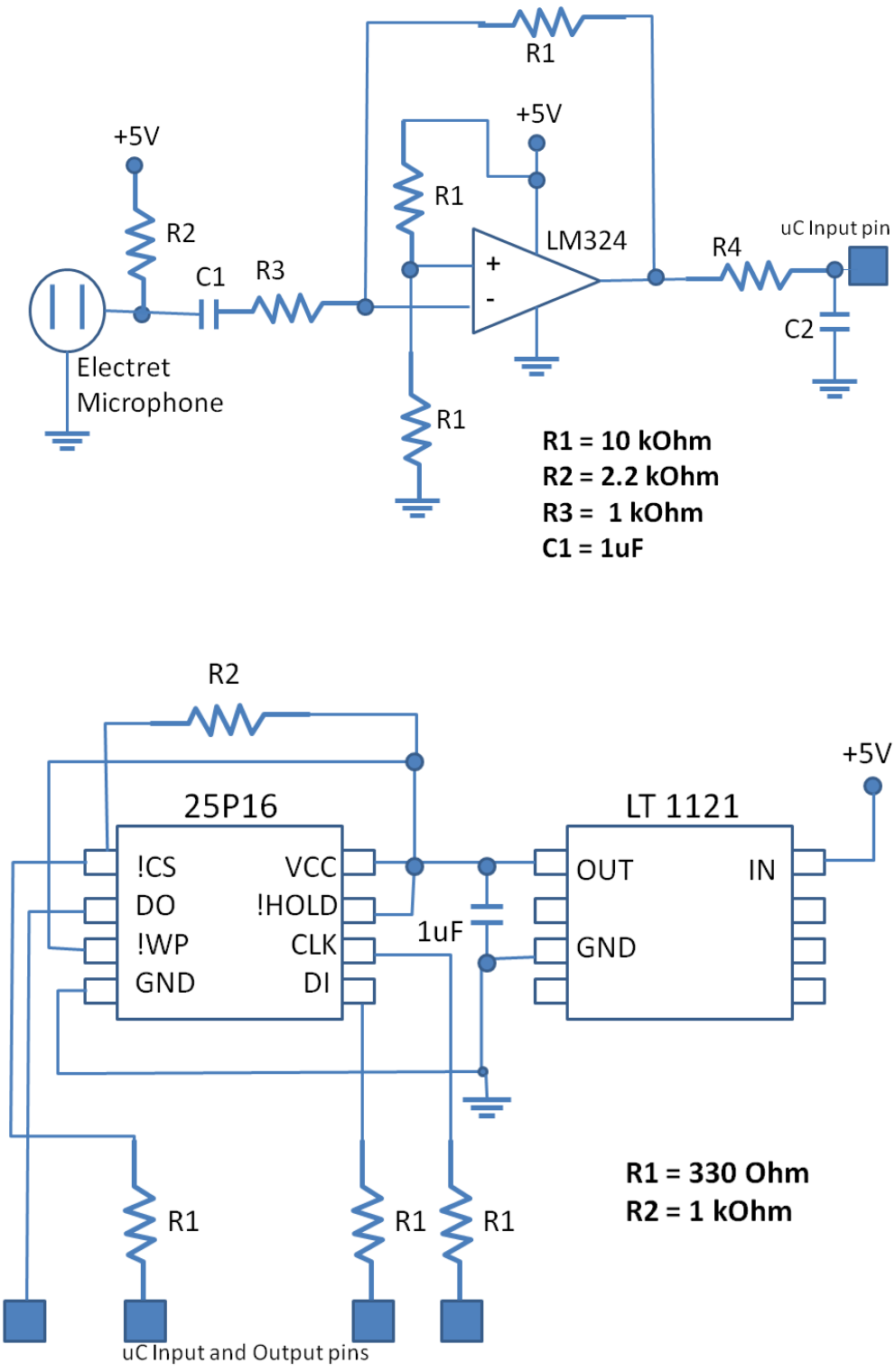
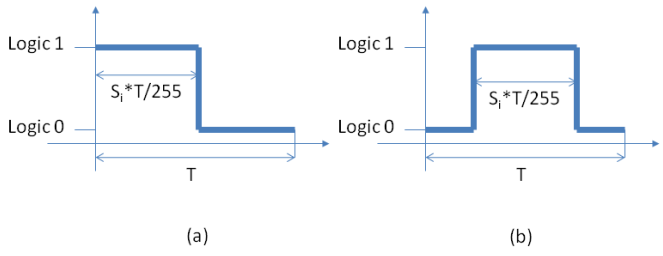


Figure 1: Circuit template available for use by students during the project.

For the software development, MPLAB includes many built-in functions to simplify the software development when using processor peripherals, however this is usually not encouraged by the instructor in an effort to instead encourage the students to consider the specific registers involved and to understand the functionality of the peripherals at the register level. The memory device is interfaced using the SPI (Serial Peripheral Interface) protocol. The specifications and functionality of this protocol is covered in the embedded systems course and the target processor does include the SPI protocol with assigned pins for three or four wire interfacing. The SPI can be configured to interface with (slave) devices in parallel or in series. The processor assigned slave select pin toggles at the beginning and end of all bytes sent through the interface. As with most applications, one size does not fit all. The memory device requires this toggling only at the beginning and end of a packet of bytes. Inter-packet toggling causes improper responses from the memory device. To get around this problem, a supplemental pin is designated as the slave select and the processor assigned slave select pin is not used. In the instructor solution, PORT C pin 2 was used (Refer to the Appendix for details). The processor facilitates the transfer of bytes through straightforward register bit toggling. The project does require the careful sequencing of the proper bytes to initiate stored data transfer.

Replaying the stored data through a speaker required the students to implement appropriate driver circuitry. Possible options include a simple filtered transistor-based driver or an amplifier circuit designed around a device such as the LA4510 power amplifier. In addition, to complete Option 2 of the project, the data is stored digitally but must be delivered to the speaker as an analog signal. Possible options include an external D/A interface (The target processor does not include an on-board D/A converter) or internal approximate D/A conversion coupled with an external filter. This could be implemented by generating a sequence of logic zeros and ones each having duration a function of the digitized sample magnitude. There are multiple functions which could be used. Two possible functions are shown in the figure below.



Digital sample, S_i, with 8-bit encoding has a range from 0-255.
Assume sample rate during recording was $f_R = 1/T$ Hz.

Figure 2: Example D/A conversion sequence.

The instructor solution utilized the method (a) from Figure 2 with each stored sample being translated into two digital bit segments. The first segment is logic one and the last segment is logic zero. The combined pulse width of the two segments taken together remains constant, but the first segment pulse width varies in proportion to the stored sample magnitude. See the Appendix for details.

Results

This project was well received for a few reasons. First, the students are challenged with a hardware circuit design which is simple but requires soldering SOIC devices. Second, the project relates to a very familiar application to the students, namely MP3 players. Third, this project integrates user I/O, external memory, flexible data storage and some data processing requiring the students to consider multiple design options for implementation. This allowed the students some flexibility in their design and presented a competitive aspect to the project.

The students found the hardware aspects of the project to be understandable, but challenging. Many student groups assumed their interface design would work "out of the box". The project did challenge their circuit diagnostics skills. The instructor did provide a basic program which could be used to test the memory device interface implemented by each student group. As with most embedded system projects, problems can surface in the hardware and software portions of the design. Isolating these problems is critical in yielding proper functionality. The students were encouraged to design their own test software for projects like this as part of a systematic approach to design verification. This project has been offered multiple times over the span of three years. The last offering utilized an alternative processor, but with the same project goals. Most student groups were at least partially successful in the completion of the project and felt the experience was valuable.

References

Microchip Technology Inc. (2004), *PIC18F6585/8585/6680/8680 Data Sheet*, Available from the www.microchip.com website.

Student versions of MPLAB IDE and MPLAB C18 compiler is available from the www.microchip.com website.

Biography

VINCENT WINSTEAD
Dr. Vincent Winstead is an associate professor in the electrical and computer engineering and technology department at Minnesota State University, Mankato. He completed his Ph.D. degree at the University of Wisconsin, Madison in Electrical Engineering.

Appendix:

```
#include <p18f8680.h>
#include "lcd.h"
#include "serial.h"

void high_isr(void);

char LF = 0x0a; // line feed
char CR = 0x0d; // carriage return
char send_WE = 0x06;  // write enable
char send_CE = 0xc7;  // chip erase
char send_WS[2] = {0x01, 0x02};  // WEL set, write status register address and data for global memory
access
char read_ID[6] = {0x90, 0x00, 0x00, 0x00, 0x00, 0x00};  // read ID address and placeholder bytes
char read_ID2[4] = {0x9f, 0x00, 0x00, 0x00};
char read_data[4] = {0x03, 0x00, 0x00, 0x00};
char read_status = 0x05;
char send_PP[14]; // page program command, address and data bytes
char PP_A1 = 0x00; // store at memory locations 0x000000 - 0x00000A
char PP_A2 = 0x00;
char PP_A3 = 0x00;

char send_data[11]="abcdefghij";
unsigned char rec_data[10];
unsigned char TEMP_info[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

unsigned char S_Rec, Temp_Rec[2], Play_loop;
int i;
unsigned long Add_Start, Num_Addresses;
unsigned int Record_Seconds=0, Record_count=0;

rom char *msg1 = "I am very happy.";
rom char *blank = "                ";
rom char *msg_complete = "Operation Complete!";
rom char *msg_page = "Page ";
rom char *msg_write = " written";
rom char *C19200 = "Connection established at 19200 baud...";
rom char *Windbond = "Winbond Serial Flash Device Detected";
rom char *P80 = "Device ID: W25P80 (8Mbit or 1MByte)";
rom char *P16 = "Device ID: W25P16 (16Mbit or 2MByte)";
rom char *EC = "Enter command (0=exit, 1=record, 2=playback, 3=display, 4=delete, 5=stats): ";
rom char *NS = "Number of seconds (1-200): ";
```

```
rom char *SA = "Starting address (hex): ";
rom char *SS = "Starting sector (decimal): ";
rom char *NA = "Number of addresses (decimal): ";
rom char *NSectors = "Number of sectors (decimal): ";
rom char *msg_recording = "Recording...";
rom char *AW = "Last Address Written: ";
rom char *PR = "Memory Remaining: ";



void Data_To_M25P80(char xc)
{
	char temp;
	PORTCbits.RC2 = 0;  // CS active low
	SSPBUF = xc;
	while (!SSPSTATbits.BF);
	temp = SSPBUF; // clear BF flag
	PORTCbits.RC2 = 1;  // end of transmission
}

void RepeatedChar_To_M25P80(char *ptr, unsigned char length, char xc, unsigned int num)
{
	char temp;
	PORTCbits.RC2 = 0;  // CS active low
	while (length)
	{
		SSPBUF = *(ptr++);
		while (!SSPSTATbits.BF);
		temp = SSPBUF; // clear BF flag
		length--;
	}
	while (num)
	{
		SSPBUF = xc;
		while (!SSPSTATbits.BF);
		temp = SSPBUF; // clear BF flag
		num--;
	}
	PORTCbits.RC2 = 1;  // end of transmission
}

void StrData_To_M25P80(char *ptr, unsigned char length)
{
	char temp;
	PORTCbits.RC2 = 0;  // CS active low
	while (length)
	{
		SSPBUF = *(ptr++);
		while (!SSPSTATbits.BF);
		temp = SSPBUF; // clear BF flag
		length--;
	}
	PORTCbits.RC2 = 1;  // end of string transmission
}

void StrData_To_From_M25P80(char *ptr, unsigned char s_len, unsigned char r_len)
```

```
{
	char temp;
	int index = 0;
	PORTCbits.RC2 = 0;  // CS active low
	while (s_len)
	{
		SSPBUF = *(ptr++);
		while (!SSPSTATbits.BF);
		temp = SSPBUF;  // clear BF flag
		s_len--;
	}
	while (r_len)
	{
		SSPBUF = 0x00;
		while (!SSPSTATbits.BF);
		rec_data[index++] = SSPBUF;
		r_len--;
	}
	rec_data[index] = '\0';
	PORTCbits.RC2 = 1;  // end of transmission
}

void Run_Record(int num_sec)
{
	unsigned char local_num;
	PORTD = num_sec;
	local_num = PORTD;
	//while(1);

	// clear memory
	Data_To_M25P80(send_WE);  // enable memory writes
	do
	{
		StrData_To_From_M25P80(&read_status, 1, 1);
	} while (rec_data[0] & 0x01);
	Data_To_M25P80(send_CE);  // chip erase
	do
	{
		StrData_To_From_M25P80(&read_status, 1, 1);
	} while (rec_data[0] & 0x01);

	// initialize
	Record_Seconds = 0;
	Record_count = 0;
	Add_Start = 0x00000A;

	// configure ADC
	ADCON0 = 0x01;  // activate ADC module with channel AD0
	ADCON1 = 0x0E;  // allow ADC on PORTA pin 0, Vref+ = Vdd, Vref- = Vss
	ADCON2 = 0x26;  // Tad = 2us, 8*Tad acquisition time, left justified

	// configure timer 0
	T0CON = 0x08;  // 1:1 prescale, 16-bit timer (8 MHz rate)
	TMR0H = 0xFC;
	TMR0L = 0x18; // overflow in 125us (8 kHz sampling)
	RCON = 0x00;  // disable priority levels
```

```
	INTCON = 0x80;  // enable interrupts, zero flags
	T0CONbits.TMR0ON = 1;  // enable timer
	INTCONbits.TMR0IE = 1;  // TMR0 interrupt
	SerialROMStringSend_wLFCR(msg_recording);

	while (Record_Seconds != local_num)
		PORTD = local_num;

	// disable timer 0 interrupt, ADC module and timer 0
	INTCONbits.TMR0IE = 0;
	ADCON0 = 0x00;
	T0CON = 0x00;

	Add_Start -= 2;
	send_PP[0] = 0x02;
	send_PP[1] = 0x00;
	send_PP[2] = 0x00;
	send_PP[3] = 0x00;
	send_PP[4] = (Add_Start >> 16) & 0xff;
	send_PP[5] = (Add_Start >> 8) & 0xff;
	send_PP[6] = Add_Start & 0xff;
	send_PP[7] = 0x00;

	do
	{
		StrData_To_From_M25P80(&read_status, 1, 1);
	} while (rec_data[0] & 0x01);
	Data_To_M25P80(send_WE);  // enable memory writes
	do
	{
		StrData_To_From_M25P80(&read_status, 1, 1);
	} while (rec_data[0] & 0x01);
	StrData_To_M25P80(&send_PP[0], 8); // fill four bytes with last address of record
}

void Run_Playback(unsigned long S_A, unsigned long N_Sec)
{
	unsigned char S_S, S_E, count2=0;
	unsigned long count1=4, last_address;

	// configure PWM (RC1)
	TRISC &= 0xFD;  // PORTC pin 1 configured as output
	T0CON = 0x08;  // 1:1 prescale, 16-bit timer (8 MHz rate)
	last_address = 4 + (N_Sec * 8000);

	// read samples from memory

	for (; count1 < last_address; count1++)
	{
		read_data[1] = (count1 >> 16) & 0xff;
		read_data[2] = (count1 >> 8) & 0xff;
		read_data[3] = count1 & 0xff;
		do
		{
			StrData_To_From_M25P80(&read_status, 1, 1);
		} while (rec_data[0] & 0x01);
```

```
                StrData_To_From_M25P80(&read_data[0], 4, 2); // read 1 byte
                //CCPR2L = rec_data[0];

                TMR0H = 0x00;
                TMR0L = 0x00;  // initialize
                T0CONbits.TMR0ON = 1;  // enable timer
                for (count2=0; count2 < 4; count2++)
                {
                        TMR0H = 0xFF;
                        TMR0L = 255 - rec_data[0];
                        INTCONbits.TMR0IF = 0;
                        PORTCbits.RC1 = 1;
                        while (!INTCONbits.TMR0IF);
                        TMR0H = 0xFF;
                        TMR0L = rec_data[0];
                        INTCONbits.TMR0IF = 0;
                        PORTCbits.RC1 = 0;
                        while (!INTCONbits.TMR0IF);
                }
                T0CONbits.TMR0ON = 0;
        }
}

void Run_Display(unsigned long S_A, unsigned long N_A)
{
        unsigned char count1, count2;
        if (N_A % 10)
                N_A -= (N_A % 10);
        for (; N_A > 0; N_A -= 10, S_A += 10)
        {
                read_data[1] = (S_A / 65536) & 0xff;
                read_data[2] = (S_A / 255) & 0xff;
                read_data[3] = S_A & 0xff;
                StrData_To_From_M25P80(&read_data[0], 4, 10); // read 10 bytes
                for (count2 = 0; count2 < 10; count2++)
                {
                        SerialStringSend(itoa((int) rec_data[count2], &TEMP_info[0]));
                        SerialCharSend(' ');
                }
                SerialCharSend(LF);
                SerialCharSend(CR);
        }
}

void Run_Delete(unsigned long S_A, unsigned long N_A)
{
        unsigned long count1, count2;
        Data_To_M25P80(send_WE);  // enable memory writes
        do
        {
                StrData_To_From_M25P80(&read_status, 1, 1);
        } while (rec_data[0] & 0x01);
        if (N_A == 32)
        {
                Data_To_M25P80(send_CE);  // chip erase
                do
```

```
                {
                        StrData_To_From_M25P80(&read_status, 1, 1);
                } while (rec_data[0] & 0x01);
                send_PP[0] = 0x02;
                send_PP[1] = 0x00;
                send_PP[2] = 0x00;
                send_PP[3] = 0x00;
                Data_To_M25P80(send_WE);  // enable memory writes
                RepeatedChar_To_M25P80(&send_PP[0], 4, 0x00, 4); // fill first four bytes with 0x00
(100%)
        }
        else
        {
                for (; N_A > 0; N_A--)
                {
                        send_PP[0] = 0xd8;  // sector erase
                        send_PP[1] = S_A & 0xff;
                        send_PP[2] = 0x00;
                        send_PP[3] = 0x00;
                        StrData_To_M25P80(&send_PP[0], 4); // fill 64k bytes with 0xff (erase)
                        S_A++;
                        do
                        {
                                StrData_To_From_M25P80(&read_status, 1, 1);
                        } while (rec_data[0] & 0x01);
                }
        }
}

void Run_Stats(void)
{
        long last_address;
        read_data[1] = 0x00;
        read_data[2] = 0x00;
        read_data[3] = 0x00;
        StrData_To_From_M25P80(&read_data[0], 4, 3); // read the last data byte address written
        last_address = ((long) rec_data[0] * 65536) + ((long) rec_data[1] * 256) + ((long) rec_data[2]);
        if (last_address > 0x1fffff)
                last_address = 0x1fffff;
        SerialROMStringSend(AW);
        SerialStringSend_wLFCR(ltoa(last_address,&TEMP_info[0]));
        SerialROMStringSend(PR);
        SerialStringSend(ltoa((long) (100-((float) last_address/0x1fffff*100)),&TEMP_info[0]));
        SerialCharSend('%');
        SerialCharSend(LF);
        SerialCharSend(CR);
}



#pragma code high_vector = 0x08
void interrupt_high(void)
{
        _asm
        GOTO high_isr
        _endasm
```

```
}
#pragma code // resume general code functions
#pragma interrupt high_isr
void high_isr(void)
{
	if (INTCONbits.TMR0IF)
	{
		TMR0H = 0xFC;
		TMR0L = 0x18;
		INTCONbits.TMR0IF = 0;  // clear flag
		ADCON0bits.GO = 1;  // start AD conversion
		Record_count++;  // increment msec counter
		if (Record_count == 10000)
		{
			Record_count = 0;
			Record_Seconds++;
			PORTD = Record_Seconds;
		}
		send_PP[0] = 0x02;
		send_PP[1] = (Add_Start >> 16) & 0xff;
		send_PP[2] = (Add_Start >> 8) & 0xff;
		send_PP[3] = Add_Start & 0xff;
		while (ADCON0bits.GO);  // wait until conversion is complete

		if (Record_count % 2)
			send_PP[4] = ADRESH;
		else
		{
			send_PP[5] = ADRESH;
			do
			{
				StrData_To_From_M25P80(&read_status, 1, 1);
			} while (rec_data[0] & 0x01);
			Data_To_M25P80(send_WE);  // enable memory writes
			StrData_To_M25P80(&send_PP[0], 6); // fill two bytes with ADC results
			Add_Start += 2;
		}
	}
}

void main(void)
{
	unsigned int count1, count2;
	TRISB |= 0x01;
	TRISD = 0x00;
	TRISA |= 0x01;  // ADC input on PORTA pin 0

	// Test the LCD interface
	ADCON1 = 0x0E;		// configure PortA pins for digital
	openLCD();
	cmd2LCD(0x80);		// set cursor to column 1 row 1
	putromS2LCD(msg1);

	TRISC &= 0xD7; // configure SPI pins
	TRISC |= 0x10;
	SSPCON1 = 0x20;  // 8MHz comm. rate
```

```
SSPSTAT = 0x40;  // CKE, CKP = 1,0

TRISC &= 0xFB;  // PORTC pin 2 output (active low chip select)

PORTCbits.RC2 = 0;
PORTCbits.RC2 = 1; // sequence to reset the interface

StrData_To_From_M25P80(&read_ID[0], 4, 2);  // read Manufacturer and Device ID
//StrData_To_From_M25P80(&read_ID2[0], 1, 3);  // read Manufacturer and Device ID
cmd2LCD(0x80);
putromS2LCD(blank);
while (PORTBbits.RB0);
cmd2LCD(0x80);
for (count1=0; count1 < 2; count1++)
{
        itoa((int) rec_data[count1],&TEMP_info[0]);
        putramS2LCD(&TEMP_info[0]);
        putc2LCD(' ');
}

SerialConfig(0x24,103);
for (count1=0; count1 < 30; count1++)
{
        SerialCharSend(LF);
        SerialCharSend(CR);
}
SerialROMStringSend_wLFCR(C19200);

if (rec_data[0] == 0xef)
        SerialROMStringSend_wLFCR(Windbond);
if (rec_data[1] == 0x13)
        SerialROMStringSend_wLFCR(P80);
if (rec_data[1] == 0x14)
        SerialROMStringSend_wLFCR(P16);

StrData_To_M25P80(&send_WS[0], 2);  // command to write status register to allow global
access
Data_To_M25P80(send_WE);  // enable memory writes
StrData_To_From_M25P80(&read_status, 1, 1);
PORTD = rec_data[0];

do
{
        StrData_To_From_M25P80(&read_status, 1, 1);
        PORTD = rec_data[0];
        SerialROMStringSend(EC);
        S_Rec = RCREG;  // clear receive buffer so LFs do not accumulate
        SerialStrReceive(&TEMP_info[0]);
        S_Rec = TEMP_info[0];
        if (S_Rec == '1')
        {
                SerialROMStringSend(NS);
                SerialStrReceive(&TEMP_info[0]);
                Run_Record(atoi(&TEMP_info[0]));
        }
        else if (S_Rec == '2')
```

```
		{
			SerialROMStringSend(SA);
			SerialStrReceive(&TEMP_info[0]);
			Add_Start = num_to_long(&TEMP_info[0], 16);  // hex conversion
			SerialROMStringSend(NS);
			SerialStrReceive(&TEMP_info[0]);
			Num_Addresses = num_to_long(&TEMP_info[0], 10);  // decimal conversion
			Run_Playback(Add_Start,Num_Addresses);
		}
		else if (S_Rec == '3')
		{
			SerialROMStringSend(SA);
			SerialStrReceive(&TEMP_info[0]);
			Add_Start = num_to_long(&TEMP_info[0], 16);  // hex conversion
			SerialROMStringSend(NA);
			SerialStrReceive(&TEMP_info[0]);
			Num_Addresses = num_to_long(&TEMP_info[0], 10);  // decimal conversion
			Run_Display(Add_Start, Num_Addresses);
		}
		else if (S_Rec == '4')
		{
			SerialROMStringSend(SS);
			SerialStrReceive(&TEMP_info[0]);
			Add_Start = num_to_long(&TEMP_info[0], 16);  // hex conversion
			SerialROMStringSend(NSectors);
			SerialStrReceive(&TEMP_info[0]);
			Num_Addresses = num_to_long(&TEMP_info[0], 10);  // decimal conversion
			Run_Delete(Add_Start, Num_Addresses);
		}
		else if (S_Rec == '5')
			Run_Stats();
} while (S_Rec != '0');

while (1);



send_PP[0] = 0x02;  // page program command
send_PP[1] = PP_A1; // data address (MSByte)
send_PP[2] = PP_A2;
send_PP[3] = PP_A3; // LSByte
//for (count1=0; count1 < 10; count1++)
//		send_PP[4+count1] = send_data[count1];
//StrData_To_M25P80(&send_PP[0], 14);  // send data to the memory locations

for (count2=0; count2 < 8192; count2++)
{
		send_PP[1] = (unsigned char) (count2 >> 8);
		send_PP[2] = (unsigned char) (count2 & 0x00FF);
		send_PP[3] = 0x00;
		RepeatedChar_To_M25P80(&send_PP[0], 4, 0x55, 256);
		cmd2LCD(0xC0);
		putromS2LCD(blank);
		cmd2LCD(0xC0);
		putromS2LCD(msg_page);
		itoa((int) count2,&TEMP_info[0]);
```

```
                putramS2LCD(&TEMP_info[0]);
                putromS2LCD(msg_write);
        }
        cmd2LCD(0xC0);
        putromS2LCD(blank);
        cmd2LCD(0xC0);
        putromS2LCD(msg_complete);
        while(1);
}
```