ECE 331

Project Report

To: Dr. Song

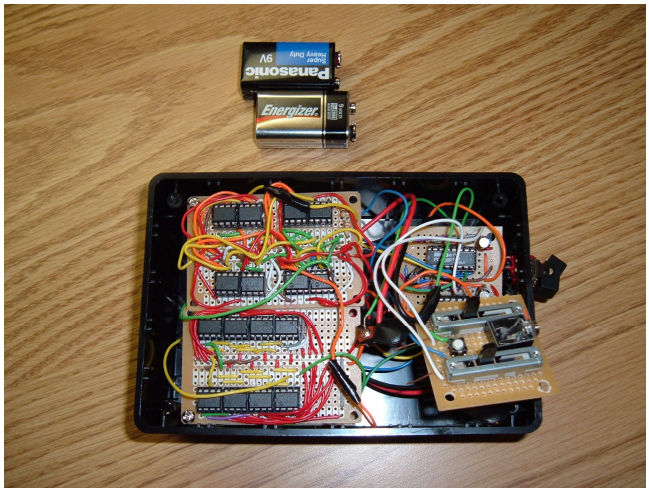
From: Curtis Rhodes and Matthew Streicher

Date: May 26, 2005

Project: PIC based headphones

Introduction:

For our ECE 331 project, we created a PIC based audio player that reads from EEPROM memory.



Tasks and priorities completed:

1. Created a Matlab program to create the audio files in the correct format (0x8000→0xFFFF for -1≤sample's value<0, and 0x0000→0x7FFF for 0≤sample's value≤1). This information was then sent to the EEPROMs using HyperTerminal and SCI.
2. Programmed an interface between SCI and I^2C so we could send a text file through SCI and put it on the EEPROM memory chips.
3. Ordered and received all the parts we needed.
4. Laid out all the plans for the final packaging.
5. Completed the interface and hardware for the DACs.
6. Finished the sequential memory read to achieve 16 kHz audio sampling.
7. Designed the PC board layout for the memory chips and DACs.
8. Completed the PC board design soldering and packaging.

Project Description:

Standard wav files store samples between the range of -1 and 1. Since the DACs we used required two's complement representation of these samples, we created a Matlab program to complete this

conversion. Our main Matlab program takes in a wav file, where it converts it to eight left and right channel text files. Each of these files has the data for about two seconds (at 16-bit 16 kHz sampling) of the input audio file. After these text files have been created, we send them to the PICs using hyperTerminal. After the audio data is on the memory chips, we read through all eight memory chips for both the left and right channels. After each read, the data for the channel is sent to its respective serial DAC. Here it is converted into an analog voltage which we can play through any device that uses a 3.5mm input.

Bill of Materials:

Cost	Cost to us	Provided by	Part
$10.00	$5.00	ECE Instrument Room	(2) PIC16877A microcontrollers
$48	$42	Microchip	(16) 24FC512 512kbit memory chips
$5.00	$5.00	ECE Instrument Room/Radio Shack	Various resistors, capacitors, buttons, and switches
$10	$10	Radio Shack	PC Boards
$5.00	$0.00	Already owned	(2) 9V batteries
$5.00	$5.00	Radio Shack	6x4x2'' Enclosure
$21.00	$0.00	Microchip	(2) PCM56P 16-bit DACs
$104.00	***$67.00***	***Total***	

Attachments:

A: Matlab code

B: C code

Memory Usage: 0.366 KB /2 KB = 18.3 %

C: Hardware Schematics

D: Software Flowcharts

Attachment A1: Channel_File_Creator2.m

```
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Channel_File_Creator2.m
% This program creates audio text files that can be
% used to program 24FC512 memory chips using
% "memoryProgrammer.c"
%
% The files can be sent to an 877A PIC Microcontroller
% using hyperterminal.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%
% The audio files created are the right and left channels
% of the desired wav file (8 chips per channel with 24FC512
% chips will produce about 16 seconds of audio at 16-bit,
% 16kHz).
%
% The audio data created is in 16-bit 2's complement
% format, which is used for a TI PCM56P Serial Digital
% to Analog Converter.
%%%%%

%%%%%%MUST BE SET TO DESIRED PATH OF OUTPUT FILES%%%%%%
Home_Path = 'C:\Documents and Settings\rhodesca\Desktop\Schoolwork\ECE
331\Final_Project\Matlab Code\';
kilobits_on_chip = 512;
bits_per_sample = 16;
samples_per_chip = kilobits_on_chip*1024/bits_per_sample;

%%%%%%ASSUMES WAV IS IN CURRENT MATLAB DIRECTORY%%%%%%
[y,fs] = wavread('Pick Up The Pieces.wav');

disp('Writing Right Channel...');

fid = fopen([Home_Path, 'Right_Audio0.txt'],'w');
for i=1:samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 0 Complete.');

fid = fopen([Home_Path, 'Right_Audio1.txt'],'w');
```

```
for i=(samples_per_chip+1):2*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 1 Complete.');

fid = fopen([Home_Path, 'Right_Audio2.txt'],'w');
for i=(2*samples_per_chip+1):3*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 2 Complete.');

fid = fopen([Home_Path, 'Right_Audio3.txt'],'w');
for i=(3*samples_per_chip+1):4*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 3 Complete.');

fid = fopen([Home_Path, 'Right_Audio4.txt'],'w');
for i=(4*samples_per_chip+1):5*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
```

```
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 4 Complete.');

fid = fopen([Home_Path, 'Right_Audio5.txt'],'w');
for i=(5*samples_per_chip+1):6*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 5 Complete.');

fid = fopen([Home_Path, 'Right_Audio6.txt'],'w');
for i=(6*samples_per_chip+1):7*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 6 Complete.');

fid = fopen([Home_Path, 'Right_Audio7.txt'],'w');
for i=(7*samples_per_chip+1):8*samples_per_chip
    hexAudio = getHexAudio(y(i,1));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
```

```
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Right Chip 7 Complete.');

disp('Finished Writing Right Channel');

disp('Writing Left Channel...');

fid = fopen([Home_Path, 'Left_Audio0.txt'],'w');
for i=1:samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 0 Complete.');

fid = fopen([Home_Path, 'Left_Audio1.txt'],'w');
for i=(samples_per_chip+1):2*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 1 Complete.');

fid = fopen([Home_Path, 'Left_Audio2.txt'],'w');
for i=(2*samples_per_chip+1):3*samples_per_chip
```

```
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 2 Complete.');

fid = fopen([Home_Path, 'Left_Audio3.txt'],'w');
for i=(3*samples_per_chip+1):4*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 3 Complete.');

fid = fopen([Home_Path, 'Left_Audio4.txt'],'w');
for i=(4*samples_per_chip+1):5*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 4 Complete.');

fid = fopen([Home_Path, 'Left_Audio5.txt'],'w');
for i=(5*samples_per_chip+1):6*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
```

```
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 5 Complete.');

fid = fopen([Home_Path, 'Left_Audio6.txt'],'w');
for i=(6*samples_per_chip+1):7*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 6 Complete.');

fid = fopen([Home_Path, 'Left_Audio7.txt'],'w');
for i=(7*samples_per_chip+1):8*samples_per_chip
    hexAudio = getHexAudio(y(i,2));
    hexAudioLength = size(hexAudio);
    switch(hexAudioLength(2))
        case 3
            hexAudio = ['0',hexAudio];
        case 2
            hexAudio = ['0','0',hexAudio];
        case 1
            hexAudio = ['0','0','0',hexAudio];
        otherwise
    end
    fprintf(fid,'%s',hexAudio);
end
fid = fclose(fid);
disp('Left Chip 7 Complete.');

disp('Finished Writing Left Channel');
```

Attachment A2: getHexAudio.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getHexAudio.m
% Finds the 2's complement hex representation of a
% value between -1 and 1.
%
% Returns a string containing the hex value.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function final_Hex = getHexAudio(negPosValue);

if(negPosValue<-1 | negPosValue>1 | length(negPosValue)>16)
    disp('Audio Sample Out of Range');
    return
elseif (negPosValue>=0)
    calcValue = round(32767*negPosValue);
else
    calcValue = round(32767*(negPosValue+1)+32768);
end
       final_Hex = dec2hex(calcValue);
```

Attachment B, 1A: Memory to DAC, I2C Subroutines.c

```
//Project : I2CSubroutines.c
//Date    : 5/23/2005
//Author  : Curtis Rhodes w/ help from Dr. Song's example
//Purpose : I2C Memory Chip Interface Subroutines
// Synchronous Serial Port
//            RC3 (pin 18) == SCL (Clock for Memory Chip)
//            RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz

#include <pic.h>
#include "memory_to_DAC.h"

//=====================================================================
==================================
// initializeI2C()
//=====================================================================
==================================
//Initializes the I2C interface
void initializeI2C(){
        //Initialize I2C
        SSPCON = 0b00111000;
        SSPADD = 4;
        ACKEN = 0;
        ACKDT = 1;
        SSPIF = 0;
}

//=====================================================================
==================================
// send_Start_Bit()
//=====================================================================
==================================
//Sends a start bit through the I2C communication line
void send_Start_Bit(){
        // I2C Start
        SEN = 1;
        while(SSPIF == 0);
        SSPIF = 0;
}

//=====================================================================
==================================
// send_Stop_Bit()
//=====================================================================
==================================
//Sends a stop bit through the I2C communication line
void send_Stop_Bit(){
        //I2C STOP
        PEN = 1;
        while(SSPIF == 0);
        SSPIF = 0;
```

```
}

//====================================================================
==================================
// acknowledge()
//====================================================================
==================================
//Sends a stop bit through the I2C communication line
void acknowledge(){
	//I2C STOP
	ACKDT = 0;
	ACKEN = 1;
	while(ACKEN == 1) {}
	while(SSPIF == 0);
	SSPIF = 0;
}

//====================================================================
==================================
// write_Data()
//====================================================================
==================================
//Sends a byte through the I2C communication line
void write_Data(unsigned int write_data){
	SSPBUF = write_data;
	while(ACKSTAT == 1) {}
	while(SSPIF == 0) {}
	SSPIF = 0;
}

//====================================================================
==================================
// receive_Data()
//====================================================================
==================================
//Waits for a byte sent to the PIC through the I2C communication line
void receive_Data(){
	RCEN = 1;
	while(SSPIF == 0) {}
	SSPIF == 0;
	//RCEN = 0;
}

//====================================================================
==================================
// wait_For_Memory_Write_Completion()
//====================================================================
==================================
//Waits for the memory to complete its current write cycle
void wait_For_Memory_Write_Completion(int control_byte){
	// Wait Until Device is Ready
	do{
		send_Start_Bit();
		write_Data(control_byte);
	}while(ACKSTAT == 1);
```

```
	send_Stop_Bit();
}

//=====================================================================
==================================
// memory_chip_write()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
void memory_chip_write(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low, unsigned char data){
	send_Start_Bit();
	write_Data(memory_chip);
	write_Data(address_high);
	write_Data(address_low);
	write_Data(data);
	send_Stop_Bit();
	//wait_For_Memory_Write_Completion(memory_chip);
}

//=====================================================================
==================================
// memory_chip_read()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
unsigned char memory_chip_read(unsigned char memory_chip, unsigned char
address_high, unsigned char address_low){
	unsigned char temp;
	send_Start_Bit();
	write_Data((memory_chip & 0b11111110));
	write_Data(address_high);
	write_Data(address_low);
	send_Stop_Bit();
	send_Start_Bit();
	write_Data(memory_chip);
	receive_Data();
	temp = SSPBUF;
	send_Stop_Bit();
	return temp;
}

//=====================================================================
==================================
// start_sequential_read()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
void start_sequential_read(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low){
	send_Start_Bit();
	write_Data((memory_chip & 0b11111110));
	write_Data(address_high);
	write_Data(address_low);
	send_Stop_Bit();
```

```
	//READ FROM MEMORY
	send_Start_Bit();
	write_Data(memory_chip);
}

//=====================================================================
==================================
// sequential_memory_chip_read()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
unsigned char sequential_memory_chip_read(){
	unsigned char temp;
	receive_Data();
	temp = SSPBUF;
	return temp;
}

//=====================================================================
==================================
// end_sequential_read()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
void end_sequential_read(){
	send_Stop_Bit();
}
```

Attachment B, 1B: Memory to DAC, memory_to_DAC.c

```
//Project : memory_to_DAC.c
//Date    : 5/23/2005
//Author  : Curtis Rhodes
//Purpose : Reads audio data from memory chips and
                    //          sends the information to serial DAC
// Synchronous Serial Port (for memory chip interface)
//          RC3 (pin 18) == SCL (Clock for Memory Chip)
//          RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
//Algorithm:
//                  Reads sequentially through all 8 memory chips.
//                  After each read, the information is sent to a
//                  serial DAC.
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz

#include <pic.h>
#include "memory_to_DAC.h"
#include "stdlib.h"
#include "stdio.h"
#if defined(_16F84)
__CONFIG(HS & WDTDIS & PWRTEN & MPEDIS);
#endif
#if defined(_16F877) || defined(_16F877A)
__CONFIG(HS & WDTDIS & PWRTEN & LVPDIS);
#endif
//=====================================================
// main program

unsigned char SCIword;
unsigned char hexValue1;
unsigned char hexValue2;
unsigned int i;
unsigned int read_control;
unsigned char write_data;
unsigned char address_high;
unsigned char address_low;
unsigned char data;

//Serial Clock Definitions
#define DAC_clock                   RC2
#define DAC_latch_enable RC1
#define DAC_data                    RD7

//Play Status is an LED that toggles when
// the program switches between memory chips
#define Play_Status_LED         RC0

void main(void)
{
        portInitialization();
        initializeI2C();
```

```
        address_high = 0x00;
        address_low = 0x00;                                                     //  Starts
read at first address
        read_control = MEMORY_CHIP0_READ_CONTROL;       // of first memory chip

        Play_Status_LED = 1;  // RC0 is used to
        while(1) {
                start_sequential_read(read_control,0x00,0x00);

                for(i=0; i<32768; i++) {
                        hexValue1 = sequential_memory_chip_read();
                        acknowledge();
                        hexValue2 = sequential_memory_chip_read();
                        if(i!=32767) {
                                acknowledge();
                        }
        ////////////////////////////////Clock in Serial DAC data
                        DAC_clock = 0;
                        PORTD = hexValue1;
                        DAC_latch_enable = 0;
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_latch_enable = 1;
                        DAC_clock = 1;

        ////////////////////////////////Clock in Serial DAC data
                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;
        ////////////////////////////////Clock in Serial DAC data
                        DAC_clock = 0;
                        PORTD = hexValue2;
                        DAC_clock = 1;
```

```
                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;
        ////////////////////////////////Clock in Serial DAC data
                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;

                        DAC_clock = 0;
                        PORTD = (PORTD<<1);
                        DAC_clock = 1;
                }
                Play_Status_LED = !Play_Status_LED;
                send_Stop_Bit();

                if(read_control == MEMORY_CHIP7_READ_CONTROL) {
                        // Loops back to beginning of audio
                        read_control = MEMORY_CHIP0_READ_CONTROL;
                }
                else {
                        read_control = read_control + 2;
                }
        }
}

void portInitialization() {
        TRISD = 0x00; // set PORTD as output
        TRISE = 0x00; // set PORTE as output

        TRISC2 = 0;     // RC5 is DAC clock
        TRISC1 = 0;     // RC4 is DAC latch enable

        TRISC0 = 0; //Toggled when moving to the next chip
}
```

Attachment B, 1C: Memory to DAC, memory_to_DAC.h

```
//Project : memory_to_DAC.h
//Date    : 5/23/2005
//Author  : Curtis Rhodes
//Purpose : Holds constants and definitions for the audio player
// Synchronous Serial Port
//           RC3 (pin 18) == SCL (Clock for Memory Chip)
//           RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz
//=====================================================
// define constants
//=====================================================

#define MEMORY_CHIP0_READ_CONTROL     0b10100001
#define MEMORY_CHIP1_READ_CONTROL     0b10100011
#define MEMORY_CHIP2_READ_CONTROL     0b10100101
#define MEMORY_CHIP3_READ_CONTROL     0b10100111
#define MEMORY_CHIP4_READ_CONTROL     0b10101001
#define MEMORY_CHIP5_READ_CONTROL     0b10101011
#define MEMORY_CHIP6_READ_CONTROL     0b10101101
#define MEMORY_CHIP7_READ_CONTROL     0b10101111

//=====================================================
// function definitions
//=====================================================
void rs232_initliazation(unsigned int, unsigned int);            // pass baud rate
void send_byte_to_PC(char);                                      // pass byte to be sent
void receive_byte_from_PC(char *);         // pass a pointer to get byte
void display_string(const char *, char length);         // pointer & length of a string

//I2C Subroutines
void send_Start_Bit();
void send_Stop_Bit();
void acknowledge();
void write_Data(unsigned int write_data);
void receive_Data();
void wait_For_Memory_Write_Completion(int control_byte);
void initializeI2C();
void memory_chip_write(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low, unsigned char data);
void portInitialization();
unsigned char memory_chip_read(unsigned char memory_chip, unsigned char
address_high, unsigned char address_low);
void start_sequential_read(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low);
unsigned char sequential_memory_chip_read();
void                                                             end_sequential_read();
```

Attachment B, 2A: Serial Memory Programmer, memoryProgrammer.h

```
//Project : I2CSubroutines.c
//Date    : 5/23/2005
//Author  : Curtis Rhodes with Additions from Dr. Song's SCI communication code
//Purpose : I2C Memory Chip Interface Subroutines
// Synchronous Serial Port
//            RC3 (pin 18) == SCL (Clock for Memory Chip)
//            RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz

//=====================================================
// define constants
//=====================================================
#define BAUD_RATE    2400
#define FREQUENCY  200                                    // Crystal frequency in kHz

//#define MEMORY_TRANSMIT_CONTROL 0x90
#define MEMORY_CHIP0_WRITE_CONTROL    0b10100000
#define MEMORY_CHIP1_WRITE_CONTROL    0b10100010
#define MEMORY_CHIP2_WRITE_CONTROL    0b10100100
#define MEMORY_CHIP3_WRITE_CONTROL    0b10100110
#define MEMORY_CHIP4_WRITE_CONTROL    0b10101000
#define MEMORY_CHIP5_WRITE_CONTROL    0b10101010
#define MEMORY_CHIP6_WRITE_CONTROL    0b10101100
#define MEMORY_CHIP7_WRITE_CONTROL    0b10101110
#define MEMORY_CHIP0_READ_CONTROL     0b10100001
#define MEMORY_CHIP1_READ_CONTROL     0b10100011
#define MEMORY_CHIP2_READ_CONTROL     0b10100101
#define MEMORY_CHIP3_READ_CONTROL     0b10100111

// Input pins
#define rs232_transmit_pin         RC6      // pin 25, serial transmit to PC
#define rs232_receive_pin          RC7      // pin 26, serial receive from PC

//=====================================================
// function definitions
//=====================================================
void rs232_initliazation(unsigned int, unsigned int);            // pass baud rate
void send_byte_to_PC(char);                                      // pass byte to be sent
void receive_byte_from_PC(char *);         // pass a pointer to get byte
void display_string(const char *, char length);         // pointer & length of a string

//I2C Subroutines
void send_Start_Bit();
void send_Stop_Bit();
void acknowledge();
void write_Data(unsigned int write_data);
void receive_Data();
void wait_For_Memory_Write_Completion(int control_byte);
void initializeI2C();
void memory_chip_write(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low, unsigned char data);
void portInitialization();
```

unsigned char memory_chip_read(unsigned char memory_chip, unsigned char address_high, unsigned char address_low);

Attachment B, 2B: Serial Memory Programmer, I2C Subroutines.c

```
//Project : I2CSubroutines.c
//Date    : 5/23/2005
//Author  : Curtis Rhodes w/ help from Dr. Song's example
//Purpose : I2C Memory Chip Interface Subroutines
// Synchronous Serial Port
//            RC3 (pin 18) == SCL (Clock for Memory Chip)
//            RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz

#include <pic.h>
#include "memoryProgrammer.h"

//=====================================================================
==================================
// initializeI2C()
//=====================================================================
==================================
//Initializes the I2C interface
void initializeI2C(){
        //Initialize I2C
        SSPCON = 0b00111000;
        SSPADD = 4;
        ACKEN = 0;
        ACKDT = 1;
        SSPIF = 0;
}

//=====================================================================
==================================
// send_Start_Bit()
//=====================================================================
==================================
//Sends a start bit through the I2C communication line
void send_Start_Bit(){
        // I2C Start
        SEN = 1;
        while(SSPIF == 0);
        SSPIF = 0;
}

//=====================================================================
==================================
// send_Stop_Bit()
//=====================================================================
==================================
//Sends a stop bit through the I2C communication line
void send_Stop_Bit(){
        //I2C STOP
        PEN = 1;
        while(SSPIF == 0);
        SSPIF = 0;
}
```

```
//=====================================================================
==================================
// acknowledge()
//=====================================================================
==================================
//Sends an acknowledge to the slave
void acknowledge(){
        //I2C STOP
        ACKDT = 0;
        ACKEN = 1;
        while(ACKEN == 1) {}
        while(SSPIF == 0);
        SSPIF = 0;
}

//=====================================================================
==================================
// write_Data()
//=====================================================================
==================================
//Sends a byte through the I2C communication line
void write_Data(unsigned int write_data){
        SSPBUF = write_data;
        while(ACKSTAT == 1) {}
        while(SSPIF == 0) {}
        SSPIF = 0;
}

//=====================================================================
==================================
// receive_Data()
//=====================================================================
==================================
//Waits for a byte sent to the PIC through the I2C communication line
void receive_Data(){
        RCEN = 1;
        while(SSPIF == 0) {}
        SSPIF == 0;
        //RCEN = 0;
}

//=====================================================================
==================================
// wait_For_Memory_Write_Completion()
//=====================================================================
==================================
//Waits for the memory to complete its current write cycle
void wait_For_Memory_Write_Completion(int control_byte){
        // Wait Until Device is Ready
        do{
                send_Start_Bit();
                write_Data(control_byte);
        }while(ACKSTAT == 1);
        send_Stop_Bit();
```

```
}

//=====================================================================
==================================
// memory_chip_write()
//=====================================================================
==================================
//Places desired data into passed memory chip / address
void memory_chip_write(unsigned char memory_chip, unsigned char address_high,
unsigned char address_low, unsigned char data){
	send_Start_Bit();
	write_Data(memory_chip);
	write_Data(address_high);
	write_Data(address_low);
	write_Data(data);
	send_Stop_Bit();
	//wait_For_Memory_Write_Completion(memory_chip);
}

//=====================================================================
==================================
// memory_chip_read()
//=====================================================================
==================================
//Reads desired data from passed memory chip / address
unsigned char memory_chip_read(unsigned char memory_chip, unsigned char
address_high, unsigned char address_low){
	unsigned char temp;
	send_Start_Bit();
	write_Data((memory_chip & 0b11111110));
	write_Data(address_high);
	write_Data(address_low);
	send_Stop_Bit();
	send_Start_Bit();
	write_Data(memory_chip);
	receive_Data();
	temp = SSPBUF;
	send_Stop_Bit();
	return temp;
}
```

Attachment B, 2C: Serial Memory Programmer, memoryProgrammer.c

```
//Project : memoryProgrammer.c
//Date    : 5/23/2005
//Author  : Curtis Rhodes with Additions from Dr. Song's SCI communication code
//Purpose : I2C Memory Chip Interface Subroutines
// Synchronous Serial Port
//           RC3 (pin 18) == SCL (Clock for Memory Chip)
//           RC4 (pin 23) == SDA (Serial Data for Memory Chip)
//
// Serial port
//           RC6 (pin 25) == TX pin to send data to PC
//           RC7 (pin 26) == RX pin to receive data from PC
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz

#include <pic.h>
#include "memoryProgrammer.h"
#include "stdlib.h"
#include "stdio.h"
#if defined(_16F84)
__CONFIG(HS & WDTDIS & PWRTEN & MPEDIS);
#endif
#if defined(_16F877) || defined(_16F877A)
__CONFIG(HS & WDTDIS & PWRTEN & LVPDIS);
#endif
//=================================================
// main program
const char       TITLE1[] = {"Serial Communication Established."};
const char       TITLE2[] = {"Please Place Front Left EEPROMs into circuit and send file."};
const char       TITLE3[] = {"Please Place Front Right EEPROMs into circuit and send file."};
const char       TITLE4[] = {"Please Place Rear Left EEPROMs into circuit and send file."};
const char       TITLE5[] = {"Please Place Rear Right EEPROMs into circuit and send file."};
const char       MESSAGE1[] = {"Chip "};
const char       MESSAGE2[] = {" completed."};

unsigned char SCIword;
unsigned char hexValue1;
unsigned char hexValue2;
unsigned int i;
unsigned int chip_write;
unsigned int write_control;
unsigned int read_control;
unsigned char write_data;
unsigned char address_high;
unsigned char address_low;
unsigned char data;

#define WRITE_MODE 1
#define READ_MODE 2
#define WRITE_AND_READ_MODE 3

#define MODE_SELECT     WRITE_AND_READ_MODE
#define NUMBER_OF_CHIPS  8
```

```
void main(void)
{
	portInitialization();
	initializeI2C();
	rs232_initliazation(BAUD_RATE, FREQUENCY);
	display_string(TITLE1, 33);
	display_string(TITLE2, 59);
	address_high = 0x00;
	address_low = 0x00;
	chip_write = 1;
	write_control = MEMORY_CHIP0_WRITE_CONTROL;
	read_control = MEMORY_CHIP0_READ_CONTROL;

	if((MODE_SELECT == WRITE_MODE)|(MODE_SELECT == WRITE_AND_READ_MODE)) {
		for(i=0; i<NUMBER_OF_CHIPS; i++) {
			while(chip_write == 1) {
				receive_byte_from_PC(&SCIword);
				if(SCIword < 0x41)
					hexValue1 = SCIword - 0x30;
				else
					hexValue1 = SCIword - 0x41 + 0x0A;
				receive_byte_from_PC(&SCIword);
				if(SCIword < 0x41)
					hexValue2 = SCIword - 0x30;
				else
					hexValue2 = SCIword - 0x41 + 0x0A;
				write_data = (hexValue1 << 4) | hexValue2;
				//PORTD = hexValue;

	memory_chip_write(write_control,address_high,address_low,write_data);
				if(address_low == 0xff) {
					if(address_high == 0xff) {
						chip_write = 0;
					}
					address_high++;
				}
				address_low++;
			}
			display_string(MESSAGE1, 5);
			display_string(MESSAGE2, 11);
			chip_write = 1;
			write_control = write_control + 2;
		}
	}

	if((MODE_SELECT == READ_MODE)|(MODE_SELECT == WRITE_AND_READ_MODE)) {
		while(1) {
			receive_byte_from_PC(&SCIword);
			if(SCIword < 0x41)
				hexValue1 = SCIword - 0x30;
			else
				hexValue1 = SCIword - 0x41 + 0x0A;
```

```
                        receive_byte_from_PC(&SCIword);
                        if(SCIword < 0x41)
                                hexValue2 = SCIword - 0x30;
                        else
                                hexValue2 = SCIword - 0x41 + 0x0A;

                        address_high = (hexValue1 << 4) | hexValue2;

                        receive_byte_from_PC(&SCIword);
                        if(SCIword < 0x41)
                                hexValue1 = SCIword - 0x30;
                        else
                                hexValue1 = SCIword - 0x41 + 0x0A;

                        receive_byte_from_PC(&SCIword);
                        if(SCIword < 0x41)
                                hexValue2 = SCIword - 0x30;
                        else
                                hexValue2 = SCIword - 0x41 + 0x0A;

                        address_low = (hexValue1 << 4) | hexValue2;

                        hexValue1                                                       =
memory_chip_read(read_control,address_high,address_low);

                        SCIword = (hexValue1 >> 4);
                        if(SCIword < 0x0A)
                                SCIword = SCIword + 0x30;
                        else
                                SCIword = SCIword - 0x0A + 0x41;
                        send_byte_to_PC(SCIword);

                        SCIword = (hexValue1 & 0b00001111);
                        if(SCIword < 0x0A)
                                SCIword = SCIword + 0x30;
                        else
                                SCIword = SCIword - 0x0A + 0x41;
                        send_byte_to_PC(SCIword);
                }
        }
}

void portInitialization() {
        TRISD = 0x00; // PORTD is output
        TRISE = 0x00; // PORTD is output
        PORTD = 0xff; // Set PORTD to ensure proper initilization
        TRISC7 = 1;     // RC7 receives data from PC
        TRISC6 = 0;     // RC6 sends data to PC
}
```

Attachment B, 2D: Serial Memory Programmer, SCISubroutines.c

```
//Project : SCISubroutines.c
//Version : 1.01
//Date    : 12/24/2002, modified 5/23/2005
//Author  : JianJian Song w/ modification by Curtis Rhodes
//Company : rose-hulman institute of technology
//Purpose : RS232 serial communicaiton routines
//Input:    serial port from PC
//Outputs:  serial output to PC
// Serial port
//            RC6 (pin 25) == TX pin to send data to PC
//            RC7 (pin 26) == RX pin to receive data from PC
//
//Chip type   : PIC16F877A
//Clock frequency     : 20 MHz
#include <pic.h>
#include "memoryProgrammer.h"
//=================================================
// display_string()
//=================================================
void display_string(const char *string, char length)
// send a string to serial port. length is string length
{
char i;
        for(i=0;i<length;i++)
                send_byte_to_PC(string[i]);
        send_byte_to_PC(0x0A);// send line feed
        send_byte_to_PC(0x0D);// send carriage return
        return;
}
// end of display_string()

//=================================================
// rs232_initliazation(baud, crystal)
//=================================================
// baud -- baud rate in # of characters per second
// crystal -- crystal speed in MHz
void rs232_initliazation(unsigned int baud, unsigned int crystal)
{
unsigned int x;  // temp variable
// transmit configuration: 8 bits, transmit enable, asynchronous mode, high speeed
//Register TXSTA = 0B10100100;          // 0B is binary data format
        TX9     = 0;                    // 8-BIT MODE
        TXEN    = 1;                    // transmit enabled
        SYNC    = 0;                    // asynchronous mode
        BRGH    = 0;                    // low speed
// receive configuration: serial port enable, 8 bits, continuous receive
// Register TXSTA = 0B10010000;
        SPEN    = 1;                    // serial port enabled
        RX9     = 0;                    // 8-bit mode
        CREN    = 1;                    // continuous receive enabled

// baud rate configuration: Baud Rate = Fosc/(16(X+1)) when BRGH=1 for high speed
// where X is the value in SPBRG
        x = 100000/baud;
```

```
        x = x*crystal;              // crystal speed in MHz
        x = (x - 64)/64;
        SPBRG = x;                  // initialize baud rate generator register
        return;
}
// end of rs232_initliazation()

//=====================================================
// send_byte_to_PC()
//=====================================================
// send one character in char to PC through serial port
void send_byte_to_PC(char letter)
{
// check transmit register status bit TRMT in register TXSTA
        while(TRMT==0) {};          // wait for previous transfer to complete
        TXREG = letter;  // send letter to PC
        return;
} //end send_byte_to_PC()
//=====================================================
// receive_byte_from_PC()
//=====================================================
// receive one character from PC through serial port
void receive_byte_from_PC(char *word)
{
// check RCIF in Register PIR1 to see if one character is received
        while(RCIF==0) {};          // wait for receive completion
        *word    = RCREG;           // get character from Register RCREG
        RCIF     = 0;               // clear receive flag
// echo receive character
        send_byte_to_PC(*word);
// check for receive error
        return;
} // end send_byte_to_PC()
```