DIGITAL FOUR TRACK RECORDER

Lucius Perreault

Andrew Macdonald

Allen Parker

March 15, 2001

ELEX 290 PROJECT COURSE

Alan Duncan
Mel Gibson
Godfried Pimlott
Joe Benge

Memorandum

To: Joe Benge
Alan Duncan
Mel Gibson
Godfried Pimlott

From: Andrew Macdonald
Allen Parker
Lucius Perreault

RE: Digital Four Track Recorder

Date: March 15, 2001

Summary:
The project proposed by our team (PMP Audio) was a digital audio recorder that can process 4 separate audio channels simultaneously. Emphasis was placed on portability, high fidelity and low cost to produce a recorder suitable for home studio enthusiasts or small scale professional applications. Due to the large amounts of data produced by high resolution analogue to digital conversion, the storage medium is a standard IDE/ATA hard-drive. As many of the controls as possible are implemented digitally in order to keep the analogue signal path as short as possible. Individual recorded tracks are to be downloadable to a PC in a known format (e.g. *.WAV) via a parallel port connection. Audio processing and hard drive interfacing will be performed on a Texas Instruments (TI) TMS320C6211 DSP run on the TI DSK6211 Demo Board.

Introduction:
Inspiration for the digital four track recorder came from the fact that there are currently few, if any, low to middle end multi-track digital recording devices available to the music industry that provide high fidelity and recording file-formats compatible with industry-standard sequencing software. Our goal was to produce a working prototype of such a device within the three months available. However due to a couple of major problems this was not possible. What has been completed to date are the following features.

- 16bit, 48kHz analogue to digital and digital to analogue conversion
- Low noise analogue signal conditioning
- Attractive packaging and case design
- Parallel port download utility

Technical Details

Analogue Front End

- 4 mono input channels. Each channel has 1 balanced input (XLR) and 1 high-impedance (1/4" phono) input.
- Input buffers use high-bandwidth, low-noise, low-offset op-amps.
- Stereo pre-amp outputs for monitors
- Stereo output for headphones

Signal Conversion

- Data conversion performed by 16 or 20 bit, 48kHz CODECs.
- CODEC driven by the DSP processor

External Data Transfer

- Parallel port running in EPP mode
- Download utility for downloading of audio tracks onto a PC

Control Panel

- Full digital control to minimize analogue signal path and thus noise level
- Controls include faders, equalizer, record functions, mute, solo and balance functions.
- Gain control to ensure proper signal presented to CODEC
- Headphones and monitor volume control

Conclusion:

It was a difficult task to design and build in three months what would normally take three professional engineers four full months of development time. However we accepted the challenged and though it is not yet fully complete it will be completed at some point in the future.

Executive Summary

The PMP Audio 4-Track Recorder is a portable unit is capable of digitally recording multiple audio sources simultaneously onto a standard hard drive. The hardware modules that make up the project are the control-panel/user-interface, the audio data acquisition system, the hard-drive interface and the parallel port for interfacing to a PC. Development has experienced some pitfalls, but the design team is committed to seeing this project to completion. Numerous improvements to the project are planned, such as a USB port, increased audio resolution and more input channels.

Contents

Introduction

This documentation is provided to describe the current internal workings of the PMP Audio digital four-track recorder. It covers all items designed and implemented by the PMP Audio team including the analogue front end, signal conversion, hard drive interface, parallel port interface and the control panel or user interface. It does not go into detail about the hard drive specifications, parallel port specifications or how to use the recorder (as that is described in the user manual).

The members of PMP Audio are:

1) Allen Parker - Analogue front end, control panel and exoskeleton.
2) Andrew Macdonald - Parallel port/USB download utility and software.
3) Lucius Perreault - Hard drive interface and data acquisition.
4) Alan Duncan - Disappointed supervisor.

This project was performed on a very small budget of less than $250.00 and within a very tight time schedule of 11 weeks for all research, development, production, testing and documentation.

Due to the amount of research that had to go into the development of the hard drive interface this was not fully completed. Also the original specification to implement a USB port into the recorder was not successful. However to implement a parallel port proved to be just as difficult but was completed. Finally because there was no hard drive implemented the recording functions on the control panel were also not implemented.

Although the PMP Audio design team did not complete this project within the eleven weeks given for the project course we plan on completing this project after graduation. We believe that this is a very marketable product and the potential for further expansion to 8, 16 or even 32 channels and more built in functionality (e.g. effects, sub-mix busses or more equalization bands) would not be difficult.

Analogue Front End

The purpose of the analogue front end is to condition the electrical audio signal into a format that the analogue to digital converter (the CODEC) can accurately convert into a digital representation of the audio signal. Three operational amplifiers perform the necessary signal conditioning which includes noise rejection, signal amplification and a DC offset. The schematic for the signal conditioning circuit can be seen in Appendix A.

In order to determine the amount of amplification required in the first two stages of the circuit I had to measure the signal level output from an industry standard microphone. Using an oscilloscope and Shure SM58 microphone, the audio signal level was measured at 20mV. To amplify this signal to the 3.3V needed by the CODEC a gain of 165 is needed. However this gain also needs to be adjustable as the signal level from the microphone will vary depending on the volume of original audio signal, but this is the maximum amplification required and it easy to turn down the gain for larger audio signals.

The second major problem that had to be overcome was the ability to deal with balanced signals. Most microphones use a balanced cable and often instruments and their pre-amplifiers use balanced cables as well. I was able to find a solution to this problem and the first problem on the Internet. Professional audio equipment manufacturer Yorkville provides full service manuals on their web site. From those manuals I was able to see how they solve these problems and implement a similar design in my solution.

Lastly the audio signal had to be offset because the CODEC could not accept signals that went below zero volts or above 3.6V, as this would destroy the CODEC. Several different methods to accomplish this were tried before the solution was found. By adding an offset to the second amplifier stage the offset was amplified as well, which is undesirable, and the clamping diodes caused heavy distortion of the signal at any level. Thus a third op-amp stage was required in order to build in the protection for the CODEC. This last stage has a unity gain (thus no amplification) and it's output has two diodes on it to clamp the signal within -0.7V to 3.0V, which is acceptable to the CODEC. Any signal larger than this is clipped but signals within this range are unaffected.

The second part of the analogue front end isn't in the front of the audio signal path but rather on the back of the audio signal path. This is the amplifier stage for the headphones output and monitor speakers output. These are required for the end user to be able to listen to the audio data that is being recorded on either headphones or monitor speakers (using an external power amplifier). For both headphones and monitor outputs the same amplifier configuration was used. This is possible only because the headphones require a much lower signal than

monitor speakers. An external power amplifier provides the rest of the amplification for the monitor speakers.

PCM3003e – CODEC Theory

A CODEC, which stands for COmpressor - DECompressor, has both an ADC and a DAC on a single IC. We chose a CODEC, the Burr Brown PCM3003e, to minimize noise and hardware.

The PCM3003e is a 16/20-Bit Single-Ended Analogue Output chip. It is hardware-configurable with a 4 pin serial interface. It accepts sampling frequencies up to 48kHz with a THD+N of –86dB. The entire chip can run off a single +3V power supply and comes in an SSOP-24 package type.

Configuration Overview

Clock Signals

The PCM3003e needs 3 clock signals for operation:

1) SCLK – This is the master clock to the CODEC. It must be 512Fs, 384Fs, or 256Fs, with Fs being the Sampling Frequency.

2) LRCIN – This is the synchronization clock that pulses at 2Fs. It determines which channel is being sampled with a HI = left channel and LO = right channel.

3) BCKIN – This is the Bit Clock. It is the rate at which data is clocked in and out of the CODEC. This Calculation determines BCKIN

BCKIN = N*Fs*(# of Channels)
N = # of bits (either 16 or 20)[1]
Fs <=48KHz
Of Channels = 2 for PCM3003

Hardware Configuration

The PCM3003e has 5 control pins for configuration. The *"all hardware"* configuration makes it easy to change the settings to many applications.

[1] Depending on value of 20BIT.

1) PDAD – Power Down Analogue to Digital - This is power control for the

2) PDDA – Power Down Digital to Analogue - This is power control for the

3) 20BIT – This controls the bit resolution used (16/20).

4) DEM1/DEM0 – De-emphasis 1/0 - These are the De-emphasize pins used to control volume levels and mode selection.[2]

Specific Configuration Settings

The PMP Audio Digital 4-track recorder utilizes 16-bit resolution sampled at 48Khz with both the De-emphasis and Power down modes disabled. Figures 1.1 and 1.2 depict the settings/signals used to obtain these settings.

PIN	**SIGNAL**
DEM1	Vcc
DEM0	GND

Figure 1.1

Pin	**SIGNAL**
PDAD	GND
PDDA	GND

Figure 1.2

[2] See Page 21 of PCM3002/3003 data sheet for complete explanation.

Problems

During this phase in the design there were two major problems to overcome.

1) Synchronization between BCLK and LRCIN must be extremely precise. There must be one transition of LRCIN[3] for every sixteen pulses of BCLK.

SOLUTION:

- To insure precise synchronization of BCLK to LRCIN make them dependent on each other. Setup Tout0 on DSP (Timer0 Output) to accept an external clock source and feed BCLK to Tinp0 (Timer0 Input).

EFFECT:

- This will solve any synchronization errors that may have occurred. Even if BCLK is noisy the phase relation will never degrade because they are dependent on each other.

2) The DSP is unable to provide a SCLK to the CODEC due to timing constraints on the DSP.

SOLUTION:

- There are several feasible solutions to this problem. We chose to use a crystal oscillator circuit.

EFFECT:

- The SCLK is completely separate from the DSP; therefor it is no longer reliant.

Results

Once the configuration is accomplished and all clock signals have been provided the quality of audio sampled by the PCM3003e is very good. With the ADC and DAC running and no signal there is approximately 10mV of noise before signal conditioning.

[3] PGT or NGT depending upon channel being sampled.

Hard Disk Drive Theory

ATA-ATAPI Standards

The ATA-ATAPI Standards[4], developed in 1989, are used heavily when interfacing to Hard Disk Drives. The information contained in these documents cover such topics as:

1) Physical and Electrical Requirements.
2) Signal Summaries and Descriptions.
3) General Operational Requirements (i.e., command delivery, register reading).
4) Register Definitions and Descriptions.
5) Command Descriptions.
6) Protocols and Timing.

File Allocation Tables

Microsoft developed the FAT[5] system in the late 1970's as a solution for file allocation in MS-DOS. Originally FAT was designed for floppy drives with less than 500k of space, but over time it has been enhanced to support larger and larger media types.

There are three FAT file system types:
1) FAT12
2) FAT16
3) FAT32.

The difference between each type is the size of the entries. FAT12 has 12 bit entries, FAT16 has 16 bit entries, and FAT32 has 32 bit entries.

Information covered in "Microsoft Extensible Firmware initiative FAT32 File System Specification" includes such topics as
1) Data Structures,
2) Type Determination,
3) Volume Initialization,
4) Directory Structures and
5) Naming Conventions.

[4] See Information Technology - AT Attachment with Packet Interface Extension (ATA/ATAPI-4) document
[5] FAT-File Allocation Table

DMA Command Protocol

We choose the DMA Command Protocol over PIO Control Protocol for two reasons:

1) DMA control bypasses use of the CPU therefore freeing it up to deal with other functions.
2) The TMS320C6211 DSK board has an onboard DMA controller for ease of interfacing to external memory or storage.[6]

Signal Interfacing to DSK board

J1	40-pin Interface HD	J1	40-pin Interface HD
Pin #	Pin # DD[0:15]		RESET -1
70	17		DASP-39
69	15		DMACK-29
68	13		DMARQ-21
67	11		INTRQ-31
66	9		HDMARDY-25
65	7		DDMARDY-27
64	5		DIOW-23
63	3		PDIAG-34
60	4		GROUND - 2,19,22,24,26,30,40
59	6		
58	8		Pin # DA[0:2]
57	10	26	35
56	12	25	33
55	14	24	36
54	16	23	CS0-37
53	18	20	CS1-38

Problems

Many problems plagued the HDD module.

[6] See section labeled “DSK6211 Demo Board” for complete overview.

The biggest obstacle to overcome was learning how to interface the HDD to the DSP board. The DSP board has dedicated External Memory Interface (EMIF) lines that are not configured for use with a hard drive. If the project were to use PIO protocol rather than DMA protocol the interfacing would be rather simple.

SOLUTION

- Use a familiar microprocessor. This will cut down research time of the processor and give more time to research important modules.

RESULTS

Because of time constraints and poor project management the hard disk drive is not yet complete.

The Parallel Port PC Interface

Summary

The PMP 4-track recorder has a parallel port interface that can be connected to PC for the purpose of downloading/uploading digital audio data to and from the recorder. Another facility available is to use the host PC as a remote control for the recorder. Commands such as volume level, EQ levels, mute, solo, play and record can be sent to the recorder via the parallel port. The port itself operates in Enhanced Parallel Port (EPP) mode to attain optimum transfer rates. The parallel port described in this section is part of the daughtercard, which interfaces to the main DSP board. At present, the parallel port interface is partially complete.

EPP Parallel Port Operation

The modern parallel port allows for bi-directional communication between the host and peripheral. There are three major operating modes of the parallel port:

SPP - Standard Parallel Port
ECP - Extended Compatibility Port
EPP - Enhanced Parallel Port

EPP mode has advantages over all other modes of operation. In EPP mode, the host arbitrates all transfers to and from the peripheral. This is a contrast to ECP where the peripheral is required to negotiate for control with the host in order to establish communication in the reverse direction. Due to the reduced protocol overhead, EPP mode can transfer data the fastest. Typical transfer rates range from 500 Kbytes/sec to 2 Mbytes/sec.

There are four different types of data transfer operations performed by an EPP port:

Data Read
Data Write
Address Read
Address Write

A data read/write operation is intended for transferring data values between the host and peripheral. Address read/write cycles are for exchanging control information. All four are used to implement a host-peripheral communication

protocol specific to the peripheral being used. It is the job of the device-driver programmer to establish this protocol.

Refer to the bibliography for a listing of the resources used in implementing the parallel port.

Peripheral-Side Hardware

There are three possibilities for implementing the main hardware functions of the recorder's parallel port.

1) A manufactured peripheral-side parallel port I/O controller IC.

Much time was spent locating and researching various parts that could have filled the design team's requirements. None were available. Given the direction that modern computer interfacing technology is progressing, it appears hardware support for parallel ports is waning.

2) A handshaking state-machine programmed on an FPGA.

Although this is the next most favourable method, there was some question as to the logic capacity of the FPGA chip and development tools available. The time required to fully research this avenue could not be afforded, so this idea was abandoned in favour of a device for which capacity was not an issue.

3) A software parallel port emulator programmed on a PIC microcontroller.

This method results in reduced data transfer rates due to the relatively slow instruction cycle time of the device being used (200ns instruction cycles on a PIC16F874 clocked at 20MHz). An advantage to using this device is its on-board parallel slave port, which can be used to interface directly to the recorder's DSP data bus.

Problem Encountered:

During a Data/Address Write cycle, the PIC puts the byte to be written to the DSP on the parallel slave port lines (direct connection to DSP's EMIF lines) BEFORE interrupting the DSP to come and get the data. This results in bus contention.

Solution:

There needs to be a way of blocking the PIC's data from the DSP's EMIF bus until the bus is free. This means that a bi-directional tristate buffer (a standard

x245 chip) must be placed between the PIC's parallel slave port and the DSP's EMIF bus. The buffer's enable input is asserted whenever the DSP signals ARE* (active LO Address Read Enable) OR AWE* (active LO Address Write Enable) are asserted. Control of the buffer's direction is controlled by the PIC.

Refer to the schematic in Appendix C

Peripheral-Side Software

The EPP port emulator program operates as follows:

The PIC polls the PC's parallel port control signals for indication of a read or write cycle.

IF a READ operation occurs, THEN

1)PIC interrupts DSP
2)Wait until DSP writes data or address byte to the PIC
3)PIC outputs byte to parallel port
4)Read cycle ends

IF a WRITE operation occurs, THEN

1) PIC takes data or address byte from the parallel port
2) PIC writes byte to DSP data bus, indicates whether its data or an address
3) PIC interrupts DSP to retrieve the byte
4) Write cycle ends

The program was written in assembly for the PIC16F874 using Microchip's MPLAB assembler/debugger software. The major reason for using assembly instead of a high-level language (like C) was to ensure the program executed as quickly as possible. Refer to Appendix G for the current version of the program.

Some problems were encountered during the development of the program.

Problem Encountered:

The PIC is interfaced to the DSP via the parallel slave port and one interrupt line, but when the DSP is interrupted to read data from the port, how does it discern whether the byte from the PIC is data or address (command) information?

Solution:

1) Have two interrupt lines from the PIC, one for read operations and one for write operations.

2) Add a ninth bit to the byte on the parallel slave port. A “0” indicates to the DSP that the 8 lower bits are data information and a “1” indicates that they are address information.

PC-Side Software

Software has been written for the host PC in order to test the operation of the peripheral-side parallel port on board the recorder. The platform chosen was QNX Real-time Platform because of its ease of programming I/O directly on absolute addresses. Windows 9x and Windows NT do not allow this and special functions that emulate device drivers are needed to perform direct I/O. Due to time constraints, QNX was the best choice.

Configuring the PC Parallel Port

The computer being used must be equipped with an EPP-capable parallel port. Most PC’s have their parallel port I/O controller mapped to the base address of 0x378. The controller contains registers located at offsets from the base address, the contents of which can be configured in order to change the behaviour of the port. The following are a list of the registers involved in operating the parallel port in EPP mode.

Table 1

Register	Address Offset	Purpose/Usage
Data Port	0x000	Data is read from & written to this location in SPP and ECP modes
Status Port	0x001	Handshaking signals from the peripheral
Control Port	0x002	Control signals to peripheral
EPP Address Port	0x003	Address (command) read/write register
EPP Data Port 1	0x004	Data read/write register for byte transfers (default)
EPP Data Port 2	0x005	Data read/write register for 16-bit transfers
EPP Data Port 3	0x006	Data read/write register for 32-bit transfers
EPP Data Port 4	0x007	Data read/write register for 32-bit transfers
Extended Control Reg.	0x402	Used to set operating modes, interrupt, DMA

Writing a value of 0x94 to the Extended Control Register (ECR) configures the I/O controller to operate in EPP mode. Once this is done, all address and data transfers are performed by reading from/writing to the Address and Data registers respectively.

The final version of the PC software will contain the following features:

- Audio data download/upload to and from the recorder and PC
- Conversion of audio data to popular formats (i.e. WAV, AIFF)
- Remote control of recorder functions from the PC over the parallel port
- Real-time status readout of the recorder's operation

Control Panel

As its name implies the control panel is where the user can control the audio signals being recorded and can even control the recording. Its main features are the liquid crystal display, long travel volume faders and the jog wheel, which replaces the large number of knobs found on other mixing consoles. As of the writing of this document the full functionality of the control panel cannot be described, as it is not yet complete.

From the control panel the user can control the

- Input signal level to the CODEC,
- Channel volume with reference to other channels,
- Equalization of the audio with high, mid range and low frequency cut and boost,
- Recording with stop, play, record, skip forward and skip back functionality and the
- Balance of each audio channel.

The user also has the ability to mute selected channels or solo selected channels in the headphones or monitors. On the display will be shown general recording and channel specific information. The process for performing any of the control functions is described fully in the user manual.

All this functionality is tied together with a PIC16F877 micro-controller. The PIC is responsible for reading the faders, buttons and jog wheel, processing the results appropriately and sending the right information to the DSP processor and the LCD. It is still to be determined the protocol that will be used for communication between the DSP and the PIC and what information will be sent. Also still to be determined is exactly what the LCD will display and how.

Conclusion

The PMP 4-Track Hard-Disk Recorder is an innovative and marketable concept. At present, the various modules that comprise the unit are still under development. Future developments will include a high-speed USB interface, 20-bit / 96kHz resolution and up to 16 input channels. Although the beginnings of this project's development were marred by setbacks and time constraints, it will be seen to completion and hopefully to market.

Bibliography

Texas Instruments DSK Board:
All documentation found at:
- http://www.ti.com

CODEC:
PCM3003 Data Sheet
DEM-PCM3003 Evaluation Fixture Data Sheet
- http://focus.ti.com/docs/prod/productfolder.jhtml?genericPartNumber=PCM3003

Hard Disk Drive:
- T13-1153D - ATA\ATAPI-4 Specifications
- FAT32 File Specification System Version 1.03, December 6, 2000

Analogue Front End:
First two audio signal conditioning stages.
- ftp://ftp.yorkville.com/pdf/servman/smap312.pdf

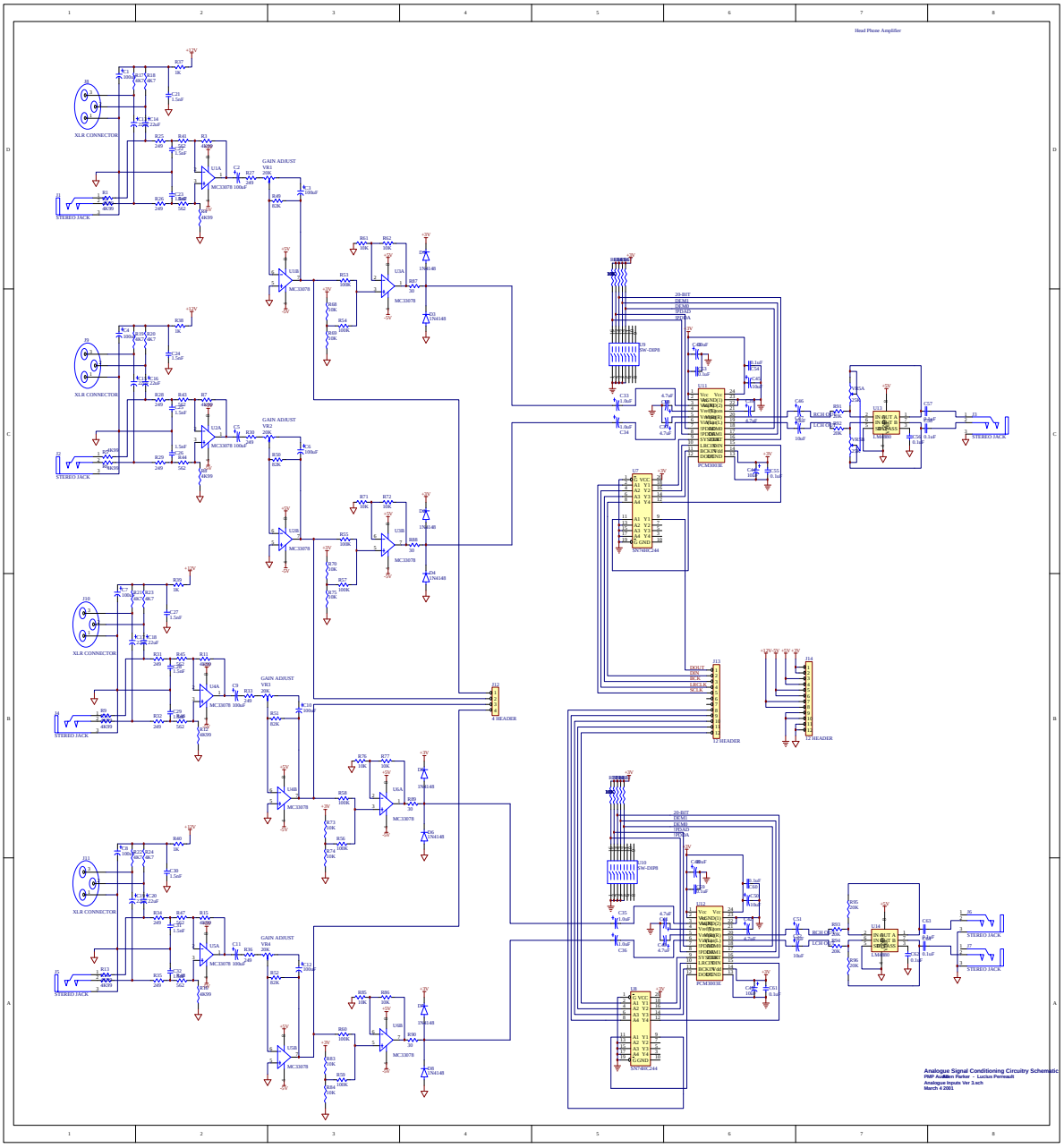
Control Panel:
LCD Interface information on Hitachi HD44780 LCD Controller
- http://www.myke.com
- http://server35.hypermart.net/shaomin/lcd/lcd0.htm

Parallel Port Interface:
- http://www.beyondlogic.org/epp/epp.htm
- http://www.lvr.com/parport.htm
- http://www.rs6000.ibm.com/resource/technology/chrpio/paral5.mak.html
- http://www.fapo.com/1284faq.htm

APPENDIX A

Analogue Front End Schematic

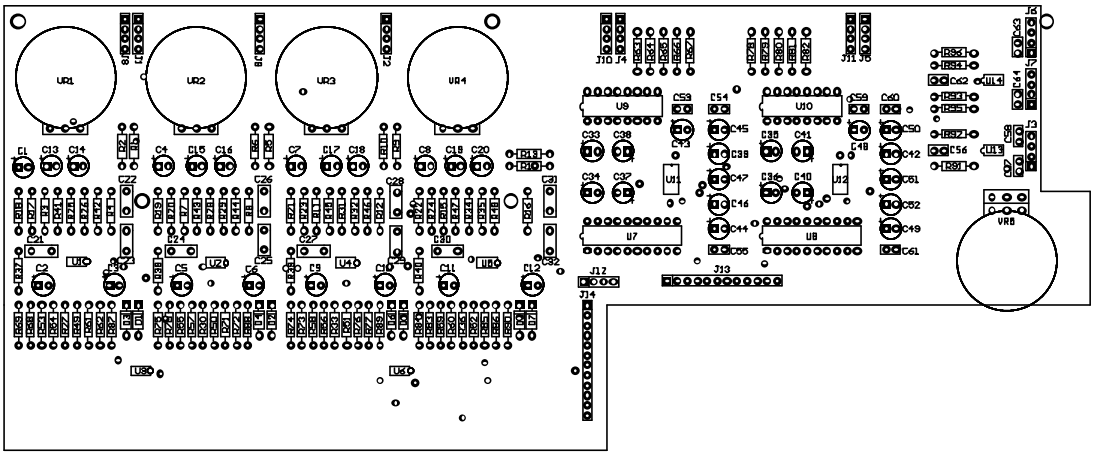
(Refer to the Protel design file Analogue Inputs Ver 3.sch for a clearer picture)



APPENDIX B

Analogue Front End Component Layout

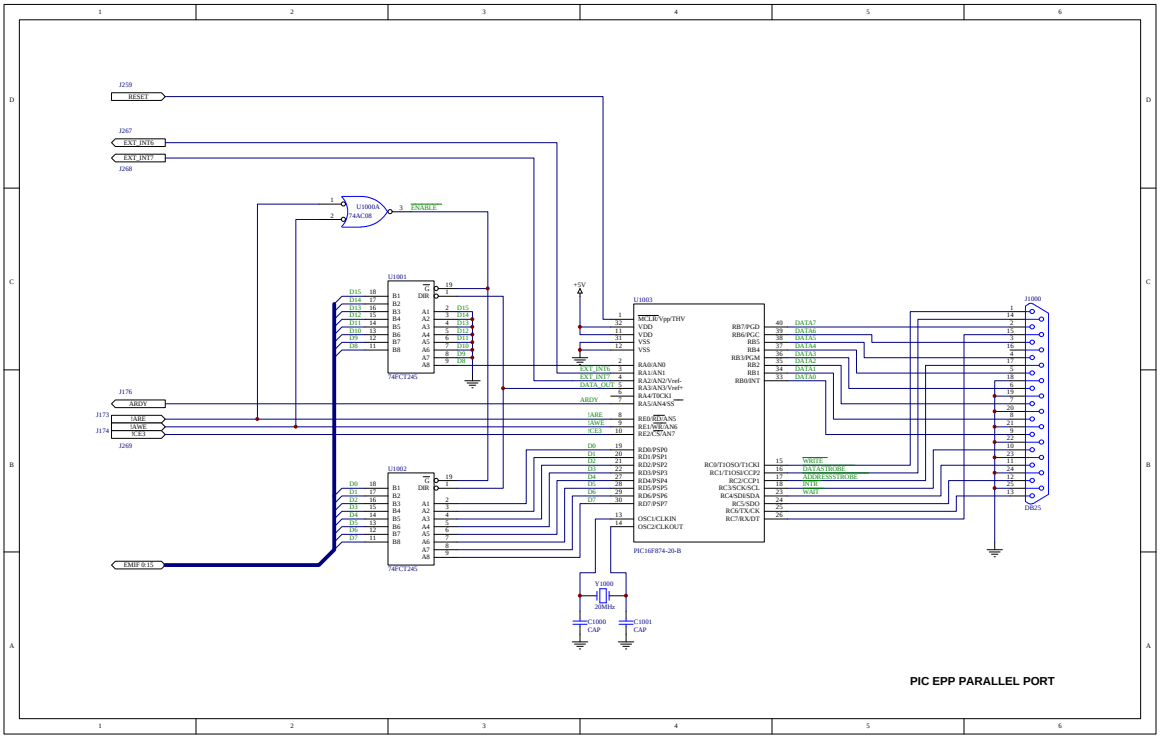
(Refer to the Protel design file Analogue Board Ver 3.PCB for a clearer picture)



APPENDIX C

Parallel Port Schematic

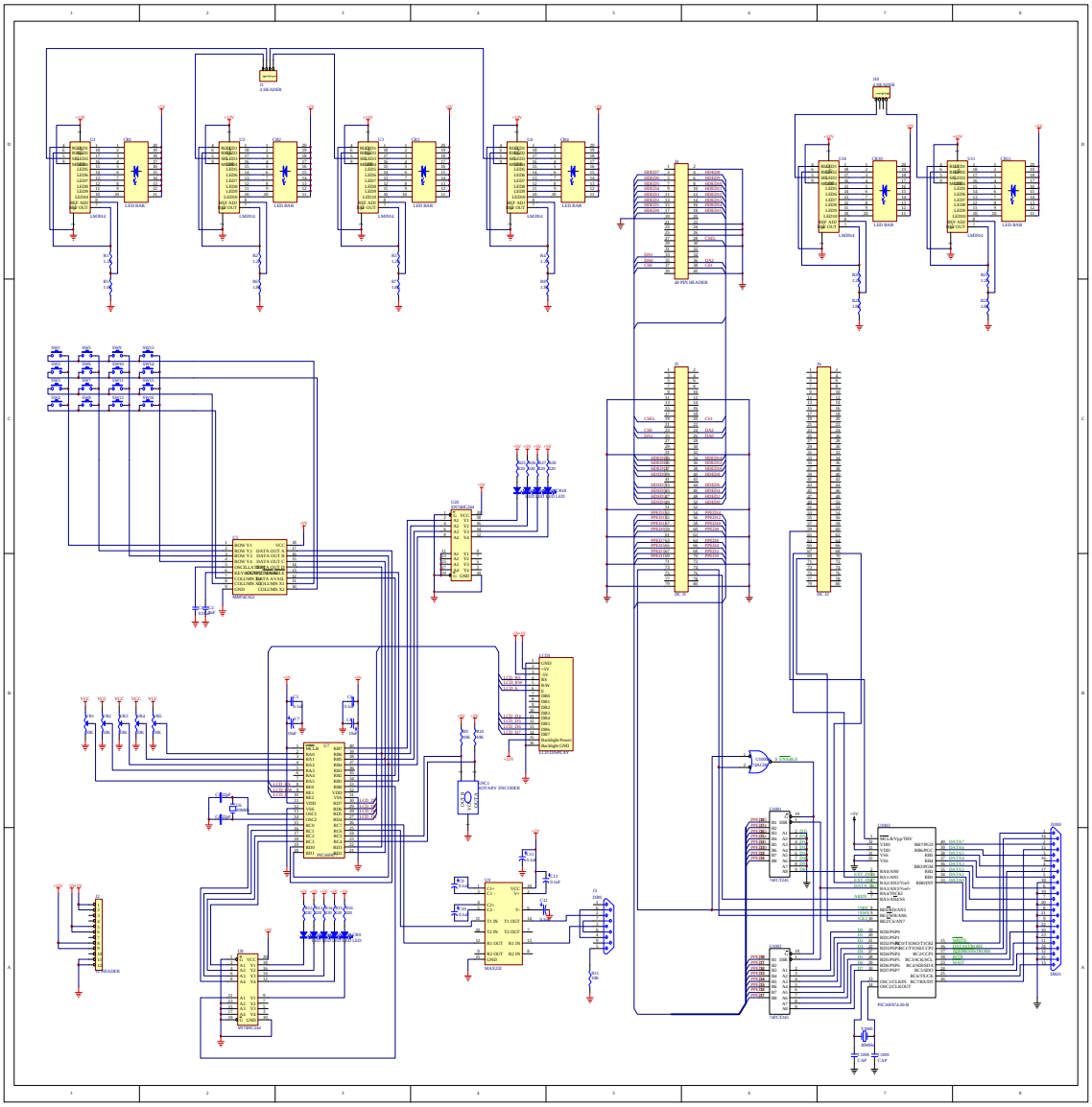
(Refer to the Protel design file Parallel Port.sch for a clearer picture)



APPENDIX D

Control Panel Schematic

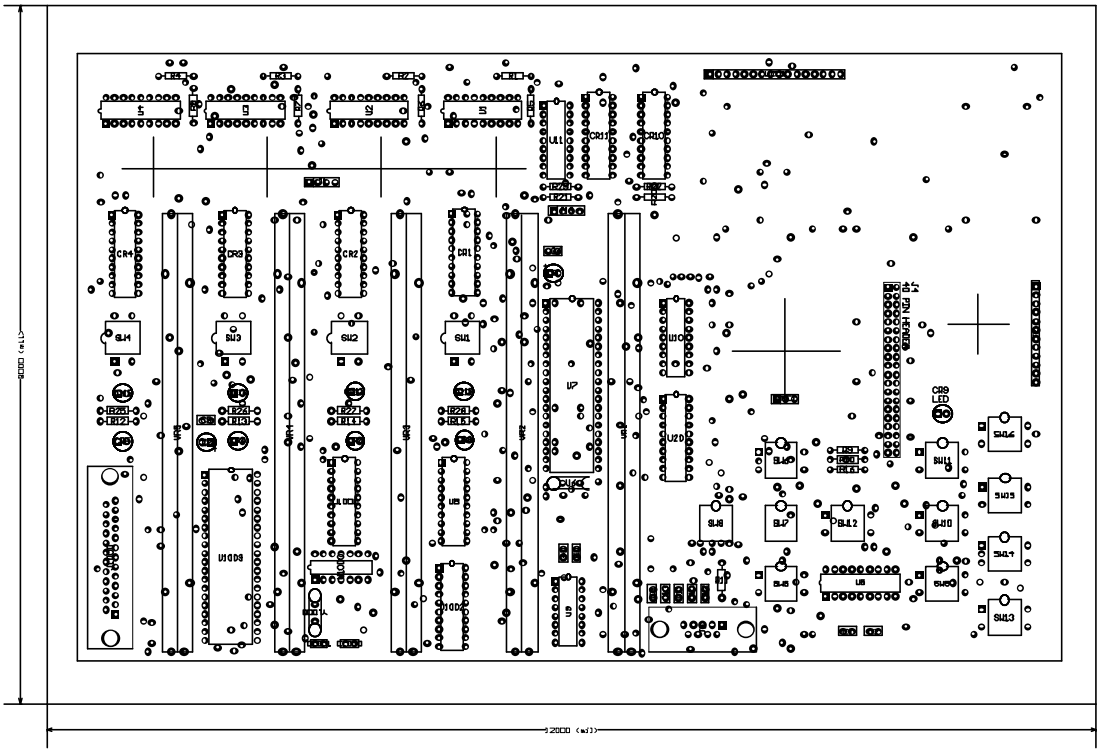
(Refer to the Protel design file Daughtercard – Control.sch for a clearer picture)



APPENDIX E

Control Panel Board Layout (incomplete)

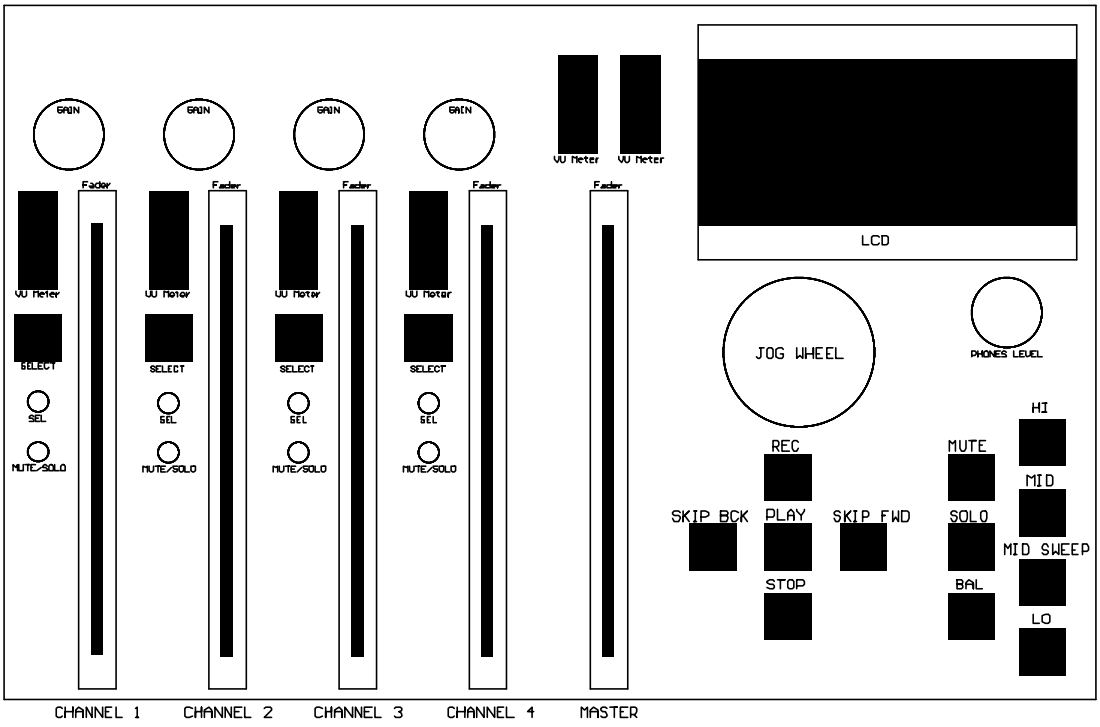
(Refer to the Protel design file Daughtercard – Control.PCB for a clearer picture)



APPENDIX F

Top Layout Of Case

(Refer to the Protel design file Mechanical Layout Top.PCB for a clearer picture)



APPENDIX G
Parallel Port Code Listing

```
/*      ============ PMPINIT.C ==============
        by Andrew Macdonald   March 11, 2001
        ---------------------------------------------------------------
        Finding information on using "devc-par" has proven
        difficult to do.  This function is used to slay devc-par
        and initialize the Winbond W83977EF    parallel-port
        controller to EPP mode when the main program starts up.
        The chip should take care of the hardware handshaking
        whenever "in8" or "out8" is used.
*/

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Toolkit headers */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* QNX headers */
#include <sys/neutrino.h>               // For thread-control access to I/O
#include <hw/inout.h>                   // For the I/O functions

/* Local headers */
#include "abimport.h"
#include "proto.h"

/* Application Options string */
const char ApOptions[] =
        AB_OPTIONS ""; /* Add your options in the "" */

/* Constants */
#define BYTE                    8
#define PPORT_BASE_ADDR  0x378  // Parallel port base address
#define ECR_OFFSET          0x402  // Offset of Extended Control Register
#define EFER1                   0x3F0  // Extended Function Enable Register location 1 (HEFRAS=0)
#define EFER2                   0x370  // Extended Function Enable Register location 2 (HEFRAS=1)
#define EFIR1                   0x3F0  // Extended Function Index Register location 1 (HEFRAS=0)
#define EFIR2                   0x370  // Extended Function Index Register location 2 (HEFRAS=1)
#define EFDR1                   0x3F1  // Extended Function Data Register location 1 (HEFRAS=0)
#define EFDR2                   0x371  // Extended Function Data Register location 2 (HEFRAS=1)
#define PPORT_DEV_NUM           0x01    // The Logical Device Number of the pport on the W83977EF


int
SetupPPort( int argc, char *argv[] )
{
        unsigned char   byte,
                        check,
                        checkmask=0x80;

        int i=0;

        // Kill the process that will interfere with this program
        system( "slay devc-par" );

        // Allow I/O
        ThreadCtl(_NTO_TCTL_IO, 0);

        //------------------- SETUP CONFIGURATION REGISTERS ----------------

        // First, enter Extended Function mode by writing 0x87 to either
```

```
        //  address 0x3F0 (HEFRAS=0) or 0x370 (HEFRAS=1) twice.
        out8( EFER1, 0x87 );
        out8( EFER1, 0x87 );

        // Write the Logical Device number to the EFIR (0x01 is the parallel port)
        out8( EFIR1, PPORT_DEV_NUM );

        // Write the same value to the EFDR
        out8( EFDR1, PPORT_DEV_NUM );

        // Set up configuration registers
        // CRF0:
        // bit7                 -       Interrupt Type (0 seems best)
        // bits(6:3)    -       ECP FIFO threshold (probably doesn't matter)
        // bits(2:0)    -       Mode (EPP 1.9 and SPP mode = 001)
        out8( EFIR1, 0xF0 );            // Select CRF0
        out8( EFDR1, 0x41 );            // Let's use 01000001

        // Set IRQ7 in CR70?
        // Set base address in CR60 & CR61 ?

        // Enable Logical Device 1
        out8( EFIR1, 0x30 );
        out8( EFDR1, 0x01 );

        // Exit Extended Function mode
        out8( EFER1, 0xAA );


        //----------------------------- SETUP ECR -----------------------------------
        // Set ECR contents for EPP mode...
        //  --Bit--                 --Function--
        //   7:5        Mode
        //    4         !ECP Error Interrupt enable
        //    3         DMA Enable
        //    2         !ECP interrupt service enable
        //    1         FIFO full
        //    0         FIFO empty
        //..We want "10010100" (actually "100101xx" --> bits 1 & 0 are read-only, so we don't care)

        byte = 0x94;
        out8( PPORT_BASE_ADDR+ECR_OFFSET, byte );




        //-------------------------------------------------------------------------------
        // ==== The port should now operate in EPP mode when "in8" and "out8" are called
        //-------------------------------------------------------------------------------

        // ...But let's make sure
        check = in8( PPORT_BASE_ADDR + ECR_OFFSET );
        if ( (check ^ 0x94) != 0 )
        {
                printf("ERROR--> Could not enter EPP mode. ECR contents: ");
                for(i=0;i<BYTE;i++)
                {
                        ( (check & checkmask) == 0 ) ? printf("0") : printf("1");
                        checkmask >>= 1;
                        checkmask &= 0x7F;
                }
                printf("\n");
        }

        /* eliminate 'unreferenced' warnings */
        argc = argc, argv = argv;

        return( Pt_CONTINUE );
}
```

```
int
Quit( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
        ThreadCtl_r(_NTO_TCTL_IO, 0);

        PtExit( EXIT_SUCCESS );

        /* eliminate 'unreferenced' warnings */
        widget = widget, apinfo = apinfo, cbinfo = cbinfo;

        return( Pt_CONTINUE );

}
/*      ============ PMPIO.C ============
        by Andrew Macdonald   March 11, 2001
        ------------------------------------------------------------
        This function is used to test the peripheral-side
        parallel port emulator (on the PIC). It performs a
        simple EPP read from the EPP Data Register.  All
        handshaking with the PIC should be taken care
        of by the PC's I/O controller chip.
*/

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Toolkit headers */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* QNX headers */
//#include <sys/neutrino.h>             // For thread-control access to I/O
#include <hw/inout.h>                   // For the I/O functions

/* Local headers */
#include "abimport.h"
#include "proto.h"

/* Constants */
#define PPORT_BASE_ADDR         0x378   // Parallel port base address
#define STATUS_OFFSET                   0x001   // Offset of SPP Status Register
#define CNTRL_OFFSET                    0X002   // Offset of SPP Control Register
#define ECR_OFFSET              0x402   // Offset of Extended Control Register
#define EPP_ADDR_OFFSET         0x003   // Offset of the Address R/W register
#define EPP_DATA_OFFSET         0x004   // Offset of the Data R/W register


int
EPPReadByte( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
        unsigned char data;
        int errorCode=0;

        data = in8( PPORT_BASE_ADDR + EPP_DATA_OFFSET );

        errorCode = PtTerminalPutc( ABW_PtTerminal_received, data );

        if( errorCode == -1) printf("Error displaying data\n");

        /* eliminate 'unreferenced' warnings */
        widget = widget, apinfo = apinfo, cbinfo = cbinfo;

        return( Pt_CONTINUE );
```

```
}
```

```
;*******************************************************************************
;********************************* PIC2PC2.ASM ***********************************
;*******************************************************************************
;       v.1.0
;       by Andrew Macdonald
;       ELEX 290 - 4-Track HD Recorder project
;-------------------------------------------------------------------------------
; This program is used for testing an EPP Read from the host PC.
;
; Part used:    PIC16F874
; Speed:        20MHz
;
; Instuctions Executed once per... (1/20MHz) * 4 = 200ns


;================================ PORT SETUP ===========================================
;       Port            Signal          Direction               used for...
;       ----            ------          -----------             ------------
;
;       PORTA           0               out                     INTERRUPT_DSP
;                       1               out                     debug LED LSB
;                       2               out                     debug LED
;                       3               out                     debug LED MSB
;                       4               --                      --
;                       5               --                      --
;
;
;       PORTB           0:7             in/out                  Port Data lines
;
;
;       PORTC                                                   (SPP)           (EPP)           (ECP)
;                       0               in                      !Strobe         !Write          HostClk
;                       1               in                      !Autofeed       !DataStrobe     HostAck
;                       2               in                      !SelectIn       !AddrStrobe     1284 Active
;                       3               out                     !Ack            !Intr           PeriphClk
;                       4               out                     Busy            !Wait           PeriphAck
;                       5               out                     PError          ---             !AckReverse
;                       6               in                                                      !ReverseRequest
;                       7               out                     !Error(!Fault)  ---             !PeriphRequest
;
;
;       PORTD           0:7             in/out                  PSP data lines (to DSP)
;
;
;       PORTE           0               in                      PSP !RD
;                       1               in                      PSP !WR
;                       2               in                      PSP !CS
;



;================================ INCLUDES ==============================================

        list    P=16F874, F=INHX8M, R=DEC
        #include p16F874.inc


;================================ CONFIGURATION BITS ====================================

        __config ( _CP_OFF & _PWRTE_ON & _XT_OSC & _WDT_OFF & _BODEN_OFF & _LVP_OFF )


;================================ CONSTANTS =============================================


;-- Configure Port C for control signals --
; SPP           EPP             ECP             PIC PORT PIN  Dirctn WRT PIC       When idle..
; ---           ---             ---             ------------  --------------      -----------
```

```
; !Strobe        !Write          HostClk         PortC:0         i               HI (SPP,ECP), either (EPP)
; !Autofeed      !DataStrobe     HostAck                 PortC:1         i               HI (SPP,EPP), LO (ECP)
; !SelectIn      !AddrStrobe     1284 Active     PortC:2         i               LO (SPP), HI (EPP,ECP)
; !Ack           !Intr           PeriphClk       PortC:3         o               HI (SPP,ECP,EPP)
; Busy           !Wait           PeriphAck       PortC:4         o               LO (SPP,EPP,ECP)
; PError ---             !AckReverse     PortC:5         o               LO (SPP), HI (ECP), either (EPP)
; Select ---             XFlag           PortC:6         o               LO (SPP), ? (ECP), either (EPP)
; !Error(!Fault)---              !PeriphRequest PortC:7          o               LO (??HI) (SPP,ECP), either (EPP)
;-------------------------------------------------------------------------------------------
; !Init          !Reset(!Init)   !ReverseRequest         << Hardwired to system RESET >>
;-------------------------------------------------------------------------------------------


; -- Control Signal Patterns --
NEG_SEQ_DETECT_PATTERN          equ     B'00000101'     ; Beginning of Negotiation sequence
SPP_IDLE_STATE                  equ     B'00001000'     ; Compatibility mode control signals idle state
EPP_IDLE_STATE                  equ     B'00001000'     ; Three LSB's are inputs (value irrelevant when writing to port)
NEG_SEQ_I_AM_COMPLIANT equ      B'11100000'     ; PError = HI, nAck = LO, nFault = HI and Select = HI
ECP_IDLE_STATE                  equ     B'10111000'     ; All 0's are inputs, so DON'T CARE


; -- Extensibility bytes --
RQ_ID_USING_BYTEMODE    equ     B'00000101'     ; Request Device ID using Byte Mode
BYTE_MODE_REV_CH        equ     B'00000001'     ; Byte Mode Reverse Channel Transfer
RQ_EPP_MODE                     equ     B'01000000'     ; Request EPP Mode


; -- Port Names --
PORTDATA                equ     PORTB
PORTCNTRL               equ     PORTC
CPUDATA                         equ     PORTD           ; PSP data lines
CPUCNTRL                equ     PORTE           ; PSP !RD, !WR and !CS


; -- Port Tristate Controls --
TRIS_PORTDATA                   equ     TRISB
TRIS_PORTCNTRL                  equ     TRISC
TRIS_CPUDATA                    equ     TRISD
TRIS_CPUCNTRL                   equ     TRISE


; -- Port Pins --
; PORTCNTRL lines:
; SPP mode control signal names
nStrobe                 equ     0
nAutofeed               equ     1
nSelectIn               equ     2
nAck                    equ     3
Busy                    equ     4
PError                  equ     5
Select                  equ     6
nFault                  equ     7
;nInit                  equ


; -- EPP Mode Control Signal Names --
nWrite                  equ     0
nDataStrobe             equ     1
nAddressStrobe                  equ     2
nIntr                   equ     3
nWait                   equ     4
Undefined1              equ     5       ; PError(SPP), !AckReverse(ECP)
Undefined2              equ     6       ; Select(SPP), XFlag(ECP)
Undefined3              equ     7       ; !Error(SPP), !PeriphRequest(ECP)


; -- ECP Mode Control Signal Names --
HostClk                 equ     0
HostAck                         equ     1
HostBusy                equ     1
```

```
ECP1284Active       equ     2
PeriphClk           equ     3
PeriphAck           equ     4
nAckReverse         equ     5
XFlag               equ     6
;nReverseRequest            equ     6
nPeriphRequest              equ     7
;XFlag              equ     5       ; On PORT A

;-- PORT A Pins --
;bit 0 connected to a pot on PICDEM board
;bit 4 connected to a switch on PICDEM board
DEBUG1                      equ     1               ; Port A debug flag (LSB)
DEBUG2                      equ     2
DEBUG3                      equ     3               ; Port A debug flag (MSB)
;XFlag              equ     5


; -- Undeclared Option Register Bit --
RBPU                equ     7



;================================ VARIABLES ===========================================

; Basic process-control flags:
EventFlags          equ     0x20            ; bank1
XMIT_READY          equ     0
NEG_ID_RQ           equ     1
NEG_EPP_RQ          equ     2
MODE                equ     3               ; SPP = LO, EPP = HI


; -- Device ID string --
DEV_ID_INDEX                equ     0x21
DEV_ID_STRING_LENGTH        equ     3
char1               equ     0x21            ; Holds device ID string "PMP"
char2               equ     0x22            ; ...that is, Parker + Macdonald + Perreault
char3               equ     0x23
devIDcharcounter    equ     0x24


; -- ISR temporary storage --
W_TEMP                      equ     0x25
STATUS_TEMP                 equ     0x26


; -- Parallel Port Buffers --
portInBuffer        equ     0x27
portOutBuffer       equ     0x28
portControlBuf      equ     0x29



;================================ MACROS ==============================================

bank0:  macro                   ; 000h - 07Fh
        bcf     STATUS,RP0
        bcf     STATUS,RP1      ; "00"
        endm

bank1:  macro                   ; 080h - 0FFh
        bsf     STATUS,RP0
        bcf     STATUS,RP1      ; "01"
        endm

bank2:  macro                   ; 100h - 17Fh
        bcf     STATUS,RP0
        bsf     STATUS,RP1      ; "10"
```

```
        endm

bank3:  macro                   ; 180h - 1FFh
        bsf     STATUS,RP0
        bsf     STATUS,RP1      ; "11"
        endm

; -- Quicker bank-change macros --

bank0to1:       macro
                bsf     STATUS,RP0
                endm

bank1to0:       macro
                bcf     STATUS,RP0
                endm



;================================ VECTORS ============================================

RESET:
        org     H'000'
        goto    Startup
;       org     H'004'
;       call    ISR

;================================ TABLES =============================================
; None

;================================ MAIN ===============================================

        org H'005'
Startup:

;       ; Port E --> PSP control lines
        bank1
;       movlw   B'00010111'             ;Port E all INPUTS, PSPMODE on
;       movwf   TRISE
;       movlw   B'10000000'             ;PSPIE on
;       movwf   PIE1
        movlw   B'00000111'             ;All A/D's turned off
        movwf   ADCON1

        clrf    TRISE
        bank0

        ; Port A --> PIC-to-DSP data bit 8, Interrupt
        bank1
        movlw   B'00000000'
        movwf   TRISA
        bank0

        clrf    PORTA
        bsf     PORTA,XFlag
        bsf     PORTA,DEBUG1        ; "001"



        ; Port B --> Parallel Port data lines
        bank1
        movlw   B'11111111'
        movwf   TRISB
        bank0

        ; Port C --> Parallel Port control lines
        bank1
        movlw   B'00000111'
        movwf   TRISC
```

```
        bank0
;       movlw   ECP_IDLE_STATE
;       movwf   PORTC

        ; Port D --> PIC Parallel Slave Port (PSP) data lines
;       movlw   B'11111111'             ;Port D all i/p's by default
;       movwf   TRISD

        bank1
        clrf    TRISD
        bank0



        ; Initialize variables
        movlw   DEV_ID_STRING_LENGTH
        movwf   devIDcharcounter
        movlw   'P'                     ; 50h, 4Dh, 50h
        movwf   char1
        movlw   'M'
        movwf   char2
        movlw   'P'
        movwf   char3

        clrf    EventFlags

;       ; Turn on Interrupts
;       bank1
;       bsf     INTCON,GIE
;       bank0

        ; Initialize Device ID string pointer
        movlw   DEV_ID_INDEX
        movwf   FSR


EPPModeIdle:
        ;Startup in EPP mode

        ; SPP IDLE EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1                    ; "010"
        bsf     PORTA,DEBUG2
        bcf     PORTA,DEBUG3

        movlw   EPP_IDLE_STATE                  ; Set control lines to idle
        movwf   PORTCNTRL

        btfss   PORTCNTRL,nWrite        ; Only send if its a READ operation
        goto    EPPModeIdle

        btfsc   PORTCNTRL,nDataStrobe       ; Send byte if LO (Address is ignored for now)
        call    SendID                  ;

        goto    EPPModeIdle


;*******************************************************************************
; FUNCTION:    SendID
;---------------------------------
; This function is responsible for sending the device ID string to the host
; via EPP Mode.  Each character is sent one at a time until the entire string
; is sent.

SendID:         ;EPP DATA READ... nWait must be LO

        ; BYTEMODE SEND-ID EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1                    ; "100"
        bcf     PORTA,DEBUG2
        bsf     PORTA,DEBUG3
```

```
        ; Send device ID using byte mode

        bank0to1
        clrf    TRIS_PORTDATA           ; Set data lines as outputs
        bank1to0

BeginTransfer:

        movf    INDF,W                  ; Put out byte on data lines
        movwf   PORTDATA
        incf    FSR,F

        bsf     PORTCNTRL,nWait         ; !Wait = HI


        btfss   PORTCNTRL,nDataStrobe   ; Wait for host to indicate receive
        goto    $-1

;DEBUG...
        bsf     PORTA,DEBUG1            ; "101"

        bank0to1
        movlw   H'FF'                   ; Set back to inputs
        movwf   TRIS_PORTDATA
        bank1to0

        bcf     PORTCNTRL,nWait         ; Assert !Wait


        movf    FSR,W                   ; Re-initialize 'PMP' string
        xorlw   DEV_ID_INDEX+2
        btfss   STATUS,Z
        goto    EndSendID
        movlw   DEV_ID_INDEX
        movwf   FSR

EndSendID:

        return


        end
```

```
;*******************************************************************************
;********************************* PICPORT.ASM *********************************
;*******************************************************************************
;       v.1.0
;       by Andrew Macdonald
;       ELEX 290 - 4-Track HD Recorder project
;-------------------------------------------------------------------------------
; This program is emulates an IEEE 1284 parallel port operating in EPP mode.
; It takes care of negotiation, read & write.
; An FPGA should have been used for this, but it's too late now!
;
; To do...
;       - Add direction control signals from Port A to '245 buffers between PSP and
;       the DSP EMIF bus (the *G signal for the buffers is provided by !ARE from
;       the DSP.  Direction control should be done all read/write routines.
;
;       - Add ARDY DSP handshaking output
;
; Part used:    PIC16F874
; Speed:        20MHz
;
; Instuctions Executed once per... (1/20MHz) * 4 = 200ns


;================================ PORT SETUP ===========================================
;       Port            Signal          Direction               used for...
;       ----            ------          -----------
;       PORTA           0               in/out                  BIT 8 (ninth bit used for operation ID)
;                       1               out                     INTERRUPT_DSP_README
;                       2               out                     INTERRUPT_DSP_WRITE2ME
;                       3               out                     -debug LED output (use for buffer direction control)
;                       4               out                     -debug LED output (use for ARDY DSP handshaking
signal)
;                       5               out                     -debug LED output (use for DSP handshaking later)
;
;
;       PORTB           0:7             in/out                  Port Data lines
;
;
;       PORTC                                                   (SPP)           (EPP)
;                       0               in                      !Strobe         !Write
;                       1               in                      !Autofeed       !DataStrobe
;                       2               in                      !SelectIn       !AddrStrobe
;                       3               out                     !Ack            !Intr
;                       4               out                     Busy            !Wait
;                       5               out                     PError          ---
;                       6               out                     Select          ---
;                       7               out                     !Error(!Fault)  ---
;
;
;       PORTD           0:7             in/out                  PSP data lines (to DSP)
;
;
;       PORTE           0               in                      PSP !RD
;                       1               in                      PSP !WR
;                       2               in                      PSP !CS
;



;================================ INCLUDES ============================================

        list    P=16F874, F=INHX8M, R=DEC
        #include p16F874.inc


;================================ CONFIGURATION BITS ==================================

        __config ( _CP_OFF & _PWRTE_ON & _XT_OSC & _WDT_OFF & _BODEN_OFF & _LVP_OFF )
```

```
;================================ CONSTANTS ========================================


;-- Configure Port C for control signals --
; SPP           EPP           ECP           PIC PORT PIN  Dirctn WRT PIC      When idle..
; ---           ---           ---           ------------  --------------      -----------
; !Strobe       !Write        HostClk       PortC:0       i                   HI (SPP,ECP), either (EPP)
; !Autofeed     !DataStrobe   HostAck       PortC:1       i                   HI (SPP,EPP), LO (ECP)
; !SelectIn     !AddrStrobe   1284 Active   PortC:2       i                   LO (SPP), HI (EPP,ECP)
; !Ack          !Intr         PeriphClk     PortC:3       o                   HI (SPP,ECP,EPP)
; Busy          !Wait         PeriphAck     PortC:4       o                   LO (SPP,EPP,ECP)
; PError ---    !AckReverse   PortC:5       o             LO (SPP), HI (ECP), either (EPP)
; Select ---    XFlag         PortC:6       o             LO (SPP), ? (ECP), either (EPP)
; !Error(!Fault)---           !PeriphRequest PortC:7      o                   LO (??HI) (SPP,ECP), either (EPP)
;-------------------------------------------------------------------------------
; !Init         !Reset  --> this signal is hardwired to the system reset
;-------------------------------------------------------------------------------


; -- Control Signal Patterns --
NEG_SEQ_DETECT_PATTERN      equ     B'00000101'     ; Beginning of Negotiation sequence
SPP_IDLE_STATE              equ     B'00001000'     ; Compatibility mode control signals idle state
EPP_IDLE_STATE              equ     B'00001110'     ; Three LSB's are inputs (value irrelevant when writing to port)
NEG_SEQ_I_AM_COMPLIANT equ  B'11100000'     ;PError = HI, nAck = LO, nFault = HI and Select = HI


; -- Extensibility bytes --
RQ_ID_USING_BYTEMODE    equ     B'00000101'     ; Request Device ID using Byte Mode
BYTE_MODE_REV_CH        equ     B'00000001'     ; Byte Mode Reverse Channel Transfer
RQ_EPP_MODE             equ     B'01000000'     ; Request EPP Mode


; -- Port Names --
PORTDATA            equ     PORTB
PORTCNTRL           equ     PORTC
CPUDATA                     equ     PORTD           ; PSP data lines
CPUCNTRL            equ     PORTE           ; PSP !RD, !WR and !CS


; -- Port Tristate Controls --
TRIS_PORTDATA               equ     TRISB
TRIS_PORTCNTRL              equ     TRISC
TRIS_CPUDATA                equ     TRISD
TRIS_CPUCNTRL               equ     TRISE


; -- Port Pins --
; PORTCNTRL lines:
; SPP mode control signal names
nStrobe             equ     0
nAutofeed           equ     1
nSelectIn           equ     2
nAck                equ     3
Busy                equ     4
PError              equ     5
Select              equ     6
nFault              equ     7
;nInit              equ


; -- EPP Mode Control Signal Names --
nWrite              equ     0
nDataStrobe         equ     1
nAddressStrobe              equ     2
nIntr               equ     3
nWait               equ     4
;
```

```
;
;


; -- Byte Mode Control Signal Names --
HostClk                 equ     0
HostAck                 equ     1
HostBusy                equ     1
_1284Active             equ     2
PeriphClk               equ     3
PeriphAck               equ     4


;-- PORT A Pins --
DATABIT8                equ     0       ; Extra data bit to DSP to identify data/address values during read
INTR_DSP_README         equ     1       ; Interrupts DSP to read from PSP
INTR_DSP_WRITE2ME       equ     2       ; Interrupts DSP to write to PSP
DEBUG1                  equ     3       ; Port A debug flag (LSB)
DEBUG2                  equ     4
DEBUG3                  equ     5       ; Port A debug flag (MSB)


; -- Undeclared Option Register Bit --
RBPU                    equ     7


;================================ VARIABLES =============

; Basic process-control flags:
EventFlags              equ     0x20            ; bank1
XMIT_READY              equ     0
NEG_ID_RQ               equ     1
NEG_EPP_RQ              equ     2
MODE                    equ     3               ; SPP = LO, EPP = HI


; -- Device ID string --
DEV_ID_INDEX            equ     0x21
DEV_ID_STRING_LENGTH    equ     3
char1                   equ     0x21            ; Holds device ID string "PMP"
char2                   equ     0x22            ; ...that is, Parker + Macdonald + Perreault
char3                   equ     0x23
devIDcharcounter        equ     0x24


; -- ISR temporary storage --
W_TEMP                  equ     0x25
STATUS_TEMP             equ     0x26


; -- Parallel Port Buffers --
portInBuffer            equ     0x27
portOutBuffer           equ     0x28
portControlBuf          equ     0x29


;================================ MACROS ===============

bank0:  macro                   ; 000h - 07Fh
        bcf     STATUS,RP0
        bcf     STATUS,RP1      ; "00"
        endm

bank1:  macro                   ; 080h - 0FFh
        bsf     STATUS,RP0
        bcf     STATUS,RP1      ; "01"
```

```
        endm

bank2:  macro                   ; 100h - 17Fh
        bcf     STATUS,RP0
        bsf     STATUS,RP1      ; "10"
        endm

bank3:  macro                   ; 180h - 1FFh
        bsf     STATUS,RP0
        bsf     STATUS,RP1      ; "11"
        endm

;================================ VECTORS ==============

RESET:
        org     H'000'
        goto    Startup
        org     H'004'
        call    ISR

;================================ TABLES ===============
; None

;================================ MAIN =================

        org H'005'

Startup:
        bank1

        ; Port E --> PSP control lines
        movlw   B'00010111'             ;Port E all INPUTS, PSPMODE on
        movwf   TRISE
        movlw   B'10000000'             ;PSPIE on
        movwf   PIE1
        movlw   B'00000111'             ;All A/D's turned off
        movwf   ADCON1

        ; Port A --> PIC-to-DSP data bit 8, Interrupt
        movlw   B'00000001'
        movwf   TRISA
        bsf     PORTA,1                 ; Set interrupt line HI

        ; Port B --> Parallel Port data lines
        movlw   B'11111111'
        movwf   TRISB

        ; Port C --> Parallel Port control lines
        movlw   B'00000111'
        movwf   TRISC
        movlw   SPP_IDLE_STATE
        movwf   PORTC

        ; Port D --> PIC Parallel Slave Port (PSP) data lines
        movlw   B'11111111'             ;Port D all i/p's by default
        movwf   TRISD

        bank0


        ; Initialize variables
        movlw   DEV_ID_STRING_LENGTH
        movwf   devIDcharcounter
        movlw   'P'
        movwf   char1
        movlw   'M'
        movwf   char2
        movlw   'P'
        movwf   char3
```

```
        clrf    EventFlags

        ; Turn on Interrupts
        bsf     INTCON,GIE

        ; Initialize Device ID string pointer
        movlw   DEV_ID_INDEX
        movwf   FSR

;*****************************************************
; To start up in EPP mode, uncomment this instruction...
; ------------------------------------------------------
        goto    EPPMode
;*****************************************************

SPPMode:
        ;Startup in compatibility mode (SPP).  This means that the
        ; port is unidirectional and therefore there is no need to
        ; identify any incoming information except for the IEEE 1284
        ; Negotiation sequence.  Wait for it and send the Device ID
        ; to the host.
        ; * Note that !ReverseRequest (!Reset in other modes) is not fed through

        ; SPP IDLE EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1                    ; "000"
        bcf     PORTA,DEBUG2
        bcf     PORTA,DEBUG3

        movlw   SPP_IDLE_STATE                  ; Set control lines to idle
        movwf   PORTCNTRL

        movf    PORTCNTRL,W                     ; Read control signals from port
        andlw   B'00000111'             ;  (examine inputs from host only)

        movwf   portControlBuf
        xorlw   NEG_SEQ_DETECT_PATTERN          ; Is it negotiation sequence?
        btfsc   STATUS,Z                ;  YES if zero
        call    NegotiationHandler

        btfsc   EventFlags,NEG_ID_RQ            ; IF the Negotiation sequence requested the Device ID,
        call    SendID                  ;  THEN send it

ModeSwitch:
        btfss   EventFlags,MODE                 ; 0 if SPP, 1 to go to EPP
        goto    SPPMode


EPPMode:
        ; Enhanced Parallel Port mode

        ; EPP-IDLE/ParseEvent EXTERNAL DEBUG FLAGS
        bsf     PORTA,DEBUG1                    ; "011"
        bsf     PORTA,DEBUG2
        bcf     PORTA,DEBUG3


        movlw   H'ff'                   ; Ensure data lines are INPUTS
        movwf   TRIS_PORTDATA

        movlw   EPP_IDLE_STATE
        movwf   PORTCNTRL

        btfss   PORTCNTRL,nDataStrobe       ; Something's happening if either !DataStrobe or !AddressStrobe
        goto    EventOccurring          ;  are asserted...
        btfss   PORTCNTRL,nAddressStrobe

EventOccurring:
        call    ParseEvent

;       movf    PORTCNTRL,W
```

```
;       movwf   portControlBuf
;       bcf     portControlBuf,0        ; Can be either read (HI) or write (LO)
;       xorlw   EPP_IDLE_STATE
;       btfss   STATUS,W
;       call    ParseEvent

        goto    EPPMode


;*******************************************************************************
; FUNCTION:   NegotiationHandler
;-----------------------------------
; Host:           - Places extensibility byte on data lines
;                 - Set nSelectIn HI and nAutofeed LO
; Peripheral:     - If peripheral is 1284 Compliant, then PError = HI, nAck = LO, nFault = HI
;                   and Select = HI
; Host:           - Drive nStrobe LO
;                 - Drive nStrobe HI and nAutofeed HI
; Peripheral:     - If the extensibility byte matches a mode that the peripheral supports, then
;                   PError = LO, nFault = LO, Select = HI
;                 - Drive nAck HI to indicate status lines are valid

NegotiationHandler:

        ; NEGOTIATION EXTERNAL DEBUG FLAGS
        bsf     PORTA,DEBUG1                ; "001"
        bcf     PORTA,DEBUG2
        bcf     PORTA,DEBUG3

        ; Extensibility byte is in "portInBuffer"
        ; There are two things that must happen:
        ;   - Request device ID
        ;   - Request EPP mode

        ; Control signals have already been recognized as beginning of NEG_SEQ
        ; Acknowledge the host:
        ;  --> PError = HI
        ;  --> !Ack   = LO
        ;  --> !Fault = HI
        ;  --> Select = HI (means that port is IEEE 1284 compliant)

        movlw   NEG_SEQ_I_AM_COMPLIANT
        movwf   PORTCNTRL

        ; Grab data from port.  Note that Host has set !Strobe low.
        ; (If this was done properly, the transition would clock the
        ;   Extensibility byte into the latch)

        movf    PORTDATA,W
        movwf   portInBuffer


CheckIDRequest:
        ; Figure out what the extensibility byte is...
        ;   - Check for ECP Device ID request:
        movf    portInBuffer,W
        xorwf   RQ_ID_USING_BYTEMODE,W
        btfss   STATUS,Z
        goto    CheckEPPRequest             ; Fall through if recognized
        bsf     EventFlags,NEG_ID_RQ        ; Device ID will be sent via Byte Mode next
        goto    Recognized

CheckEPPRequest:
        ; Device ID request not recognized, check for EPP Mode request
        movf    portInBuffer,W
        xorwf   RQ_EPP_MODE,W
        btfss   STATUS,Z
        goto    Finalize            ; Unrecognized
        bsf     EventFlags,NEG_EPP_RQ
        goto    Recognized
```

```
Recognized:
        bcf     PORTCNTRL,PError
        bcf     PORTCNTRL,nFault
;       bsf     PORTCNTRL,Select    ; Should still be HI

Finalize:
        ; Negotiation sequence is over
        bsf     PORTCNTRL,nAck          ; nACK goes HI and state of signal lines is
        btfss   EventFlags,NEG_ID_RQ    ;  compatible with the requested mode
        goto    EndNegotiation
        movlw   H'00'
        movwf   TRIS_PORTDATA

EndNegotiation:

        return



;*******************************************************************************
; FUNCTION:   SendID
;-------------------------------------
; This function is responsible for sending the device ID string to the host
; via Byte Mode.  Each character is sent one at a time until the entire string
; is sent.

SendID:

        ; BYTEMODE SEND-ID EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1            ; "010"
        bsf     PORTA,DEBUG2
        bcf     PORTA,DEBUG3


        ; Send device ID using byte mode

        clrf    TRIS_PORTDATA           ; Set data lines as outputs (again... why not?!)

BeginTransfer:
        btfsc   PORTCNTRL,HostAck  ; Wait forever until HostAck (HostBusy) goes LO
        goto    $-1                ;

        movf    INDF,W                  ; Put out byte on data lines
        movwf   PORTDATA
        incf    FSR,F

        bcf     PORTCNTRL,nAck          ; Signal host that byte is valid, sets nAck (PtrClk) LO
        nop

        btfss   PORTCNTRL,HostAck  ; Wait for host to indicate receive
        goto    $-1

        bsf     PORTCNTRL,nAck          ; Acknowledge host

        ; Host pulses as acknowledgement to the peripheral.  This program may miss it entirely...?
        btfsc   PORTCNTRL,HostClk  ; It's a pulse, therefore we just have to wait for the LO
        goto    $-1                ;  (the PGT is irrelevant)

        ; Repeat until all bytes of the Device ID are sent
        decfsz  devIDcharcounter,F
        goto    BeginTransfer

        return



;*************************************************************************
; FUNCTION:   ParseEvent
```

```
;--------------------------
; Once an EPP event has occurred, this function determines what the event
; actually is.  Valid EPP operations are as follows:
;
; EPP Data Write
; EPP Data Read
; EPP Address Write
; EPP Address Read
;
; Initial conditions:  !WRITE and (!DATASTROBE or !ADDRESSSTROBE) have been
;  pulsed.


ParseEvent:
        movf    PORTCNTRL,W             ; Sample control i/p's
        movwf   portControlBuf

        btfss   portControlBuf,nWrite
        goto    ItsARead

ItsAWrite:
        ; Determine whether or not its a DATA write or an ADDRESS write
        ;  --> Check for Data Write first

        btfsc   portControlBuf,nDataStrobe
        goto    CheckIfAddrWrite
        call    EPPDataWrite
        goto    ParseEnd

CheckIfAddrWrite:
        btfss   portControlBuf,nAddressStrobe
        call    EPPAddressWrite
        goto    ParseEnd

ItsARead:
        btfsc   portControlBuf,nDataStrobe
        goto    CheckIfAddrRead
        call    EPPDataRead
        goto    ParseEnd

CheckIfAddrRead:
        btfss   portControlBuf,nAddressStrobe
        call    EPPAddressRead

ParseEnd:
        return                          ; ANY INVALID EVENT FALLS THROUGH



;*******************************************************************************************
; FUNCTION:    EPPDataWrite
; --------------------------
; The host PC is attempting to write a data value to the port.
; 1) Read in byte from port (PC) side
; 2) Put byte on PIC's PSP (DSP data lines)
; 3) Interrupt the DSP for it to take the byte

EPPDataWrite:

        ; EPP DATA WR EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1            ; "100"
        bcf     PORTA,DEBUG2
        bsf     PORTA,DEBUG3

        movlw   H'ff'
        movwf   TRIS_PORTDATA                   ; Set port data lines as INPUTS
        bcf     OPTION_REG,RBPU                 ;   and disable pull-ups
        clrf    TRIS_CPUDATA                    ; Set data lines to DSP as OUTPUTS

        ; Tell DSP what kind of data is coming in (this must be recognized in DSP software)
```

```
        bcf     TRISA,DATABIT8                  ; Set PIC-->DSP data bit 8 as OUTPUT


        bsf     PORTCNTRL,nWait

        movf    PORTDATA,W              ; Read data parallel port data
        movwf   CPUDATA                         ;   and spit it out to the DSP
        bcf     PORTA,DATABIT8                  ; Bit8 = 0 for this transfer

        bcf     PORTA,INTR_DSP_README           ; Tell DSP that data is available
        bsf     PORTA,INTR_DSP_README           ; (DSP must know that the transferred value follows the
operation ID)


        btfss   PORTCNTRL,nDataStrobe           ; Wait for PGT from host
        goto    $-1

        bcf     PORTCNTRL,nWait

        return



;*******************************************************************************
; FUNCTION:   EPPAddressWrite
; ----------------------------
; The host PC is attempting to write a control value to the port.
; 1) Read in byte from port (PC) side
; 2) Put byte on PIC's PSP (DSP data lines)
; 3) Interrupt the DSP for it to take the byte

EPPAddressWrite:

        ; EPP ADDRESS WR EXTERNAL DEBUG FLAGS
        bsf     PORTA,DEBUG1            ; "101"
        bcf     PORTA,DEBUG2
        bsf     PORTA,DEBUG3


        movlw   H'ff'
        movwf   TRIS_PORTDATA                   ; Set data lines as INPUTS
        bcf     OPTION_REG,RBPU                 ;   and disable pull-ups
        clrf    TRIS_CPUDATA                    ; Set data lines to DSP as OUTPUTS

        ; Tell DSP what kind of data is coming in (this must be recognized in DSP software)
        bcf     TRISA,DATABIT8                  ; Set PIC-->DSP data bit 8 as OUTPUT


        bsf     PORTCNTRL,nWait

        movf    PORTDATA,W              ; Read parallel port data to be put out to the DSP
        movwf   portInBuffer            ; Store it ...
        bsf     PORTA,DATABIT8                  ; (Bit8 = 1 for this transfer)

        bcf     PORTA,INTR_DSP_README           ; Tell DSP that data is available
        bsf     PORTA,INTR_DSP_README           ; (DSP must know that the transferred value follows the
operation ID)


        btfss   PORTCNTRL,nAddressStrobe  ; Wait for PGT from host
        goto    $-1

        bcf     PORTCNTRL,nWait

        return



;*******************************************************************************
; FUNCTION:   EPPDataRead
```

```
; -------------------------
; The host PC is requesting a data byte from the PIC.
; 1) Interrupt DSP for byte, wait until byte is given
; 2) Output byte to PC port

EPPDataRead:

        ; EPP DATA RD EXTERNAL DEBUG FLAGS
        bcf     PORTA,DEBUG1                ; "110"
        bsf     PORTA,DEBUG2
        bsf     PORTA,DEBUG3


        ; Host side is tristate
        clrf    TRIS_PORTDATA                       ; Set port data lines as OUTPUTS
        bsf     OPTION_REG,RBPU                     ;   and enable pull-ups.
        movlw   H'ff'                       ; Set data lines to DSP as INPUTS
        movwf   TRIS_CPUDATA
        bsf     TRISA,DATABIT8                      ; Set Bit8 as an INPUT

        bsf     PORTCNTRL,nWait

        ; Interrupt DSP for byte to write to Parallel Port
        bcf     PORTA,INTR_DSP_WRITE2ME
        bsf     PORTA,INTR_DSP_WRITE2ME
        btfss   EventFlags,XMIT_READY
        goto    $-1

        movf    portOutBuffer,W                     ; Host PC reads data
        movwf   PORTDATA
        bcf     EventFlags,XMIT_READY

        btfss   PORTCNTRL,nDataStrobe               ; Wait for PGT
        goto    $-1

        bcf     PORTCNTRL,nWait

        return



;*******************************************************************************************
; FUNCTION:    EPPAddressRead
; ---------------------------
; The host PC is requesting a control byte from the PIC.
; 1) Interrupt DSP for byte, wait until byte is given
; 2) Output byte to PC port

EPPAddressRead:

        ; EPP ADDRESS RD EXTERNAL DEBUG FLAGS
        bsf     PORTA,DEBUG1                ; "111"
        bsf     PORTA,DEBUG2
        bsf     PORTA,DEBUG3


        ; Host side is tristate
        clrf    TRIS_PORTDATA                       ; Set port data lines as OUTPUTS
        bsf     OPTION_REG,RBPU                     ;   and enable pull-ups.
        movlw   H'ff'                       ; Set data lines to DSP as INPUTS
        movwf   TRIS_CPUDATA
        bsf     TRISA,DATABIT8                      ; Set Bit8 as an INPUT

        bsf     PORTCNTRL,nWait

        ; Interrupt DSP for byte to write to Parallel Port
        bcf     PORTA,INTR_DSP_WRITE2ME
        bsf     PORTA,INTR_DSP_WRITE2ME
        btfss   EventFlags,XMIT_READY
        goto    $-1
```

```
        movf    portOutBuffer,W                 ; Host PC reads data
        movwf   PORTDATA
        bcf     EventFlags,XMIT_READY

        btfss   PORTCNTRL,nAddressStrobe  ; Wait for PGT
        goto    $-1

        bcf     PORTCNTRL,nWait

        return


;*******************************************************************************
; Interrupt Service Routine
; -------------------------
; Something has been written to or read from the PIC's PSP by the DSP.

ISR:
        ; Save State...
        movwf   W_TEMP
        swapf   STATUS,W
        movwf   STATUS_TEMP

        bcf     PIR1,PSPIF

        ; What happened? WRITE by DSP (Host parallel port Read operation)...
        btfss   TRISE,IBF       ; =1 if PSP input buffer is full.
        goto    ReadByDSP
        movf    CPUDATA,W
        movwf   portOutBuffer
        bcf     TRISE,IBF
        bsf     EventFlags,XMIT_READY
        goto    ISREnd

ReadByDSP:
        ; ....or READ by DSP (Host parallel port Write operation)
        btfsc   TRISE,OBF       ; =0 if PSP has been read from by the DSP
        goto    ISREnd
        movf    portInBuffer,W
        movwf   CPUDATA
        bsf     TRISE,OBF

ISREnd:
        ; Restore State...
        swapf   STATUS_TEMP,W
        movwf   STATUS
        swapf   W_TEMP,F
        swapf   W_TEMP,W

        bsf     INTCON,GIE

        retfie


        end
```

APPENDIX H

CODEC Code Listing

```
#include <stdio.h>
#include <c6x.h>
#include "c6211dsk.h"
#include "cnfDSP.h"
#include "dsk6xHSB.h"

#include "lwpdefine.h"
#include "hdcommands.h"

#pragma DATA_SECTION(handShakingBuffer, "my_DataSect")
int handShakingBuffer[HS_BUFFER_LEN];

//-----------------------------------------------------------
#define  DATA_SIZE 2
#define  BLOCK_SIZE 80
#define  NUM_OF_BLOCKS 18
#define  DELAY 5

int pcm_out,pcm_in,dma_index;
int src,dst;
short *in_ptr,*out_ptr;
short in[NUM_OF_BLOCKS*BLOCK_SIZE], out[NUM_OF_BLOCKS*BLOCK_SIZE];

//-----------------------------------------------------------

short sin_table[] = {0, 23170>>2, 32767>>2, 23170>>2,0, -23170>>2, -32767>>2, -23170>>2};
int flag;

int main()
{

    /*unsigned int uiData1, uiData2;
    int iCounter = 1; */
        /* dsp and peripheral initialization */
        CSR=0x100;                              /* 100 disable all interrupts & little endian mode */
        IER=1;                      /* disable all interrupts except NMI */
        ICR=0xffff;                  /* clear all pending interrupts      */

        #if PRINT
                printf("\n*TMS320C6211/6711 DSK Confidence Test*\n");
        #endif

        ICR = 0xC000;  /*previously 0xC100*/
        IER |= 0x4102; /* previously 0x4102enable int14 timer(0),int 8 (EDMA) */
        emif_init();
        timer0_init();

   edma_init() ;

        mcbsp1_init();
        mcbsp1_test();

        /**************TURN ON****************/
   CSR|=0x1;   /*enable the interrupts*/
   /**************TURN ON****************/
        while (1) ;

}    //end of main program

/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* emif_init() - used to initialize emif                                   */
/*-------------------------------------------------------------------------*/

void emif_init()
{

        /*************************************************************************
        * Standard 6211 DSK includes 2 MT48LC1M16A1-7 devices =>4MB SDRAM       *
```

```
        *   16Mb (16-bit x 2 banks x 512K) parts = 2MB / part                   *
        *   EMIF_SDCTRL=0x07227000                                              *
        *   EMIF_SDEXT=0x54529                  Board Rev = 1                   *
        ******************************************************************************/

        *(unsigned volatile int *)EMIF_GCR = 0x3300; /* EMIF global control   */
        *(unsigned volatile int *)EMIF_CE0 = 0x30;   /* EMIF CE0control       */
        *(unsigned volatile int *)EMIF_CE1 = 0xffffff03; /* EMIF CE1 control, 8bit async */
        *(unsigned volatile int *)EMIF_CE2 = 0xffffff1f; /* EMIF CE2 control, 16bit async */
        *(unsigned volatile int *)EMIF_CE3 = 0xffffff1f; /* EMIF CE3 control, 16bit async */
        *(unsigned volatile int *)EMIF_SDCTRL = 0x07227000; /* EMIF SDRAM control     */
        *(unsigned volatile int *)EMIF_SDRP = 0x61a;       /* EMIF SDRM refresh period */
        *(unsigned volatile int *)EMIF_SDEXT = 0x54529; /* EMIF SDRAM extension    */

}

/*----------------------------------------------------------------------------*/
/*----------------------------------------------------------------------------*/
/* mcbsp1_init() - used to initialize McBSP1                                  */
/*----------------------------------------------------------------------------*/
void mcbsp0_init()
{
        volatile unsigned int temp =0;
  /* set up McBSP0                                                 */
  *(unsigned volatile int *)McBSP1_SPCR = 0;      /* reset serial port     */
  *(unsigned volatile int *)McBSP1_PCR = 0;       /* set pin control reg.  */
  *(unsigned volatile int *)McBSP1_RCR = 0x10040; /* set rx control reg.   */
                                  /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_XCR = 0x10040; /* set tx control reg.   */
                                  /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_DXR = 0;
  *(unsigned volatile int *)McBSP1_SPCR = 0x12001;/*  setup SP control reg */
  printf("in McBSP0_init\n");
}

/*----------------------------------------------------------------------------*/
/* mcbsp1_test() - used to test McBSP1                                        */
/*----------------------------------------------------------------------------*/
int mcbsp0_test()
{


        volatile unsigned int temp =0,temp1;
  /* set up McBSP0 */
  *(unsigned volatile int *)McBSP0_SPCR = 0;        /* reset serial port     */
  //*(unsigned volatile int *)McBSP0_SRGR = 0x2014004A;/* set baud rate to 1MHz */
  //*(unsigned volatile int *)McBSP0_SRGR = 0x20140002;/*set baud rate to 512(48K)*/
  //*(unsigned volatile int *)McBSP0_SRGR = 0x20140003;/*set baud rate to 384(48K)*/
  *(unsigned volatile int *)McBSP0_SRGR = 0x20140005;/*set baud rate to 256(48K)*/
  *(unsigned volatile int *)McBSP0_PCR = 0xA00;      /* set pin control reg.  */
  *(unsigned volatile int *)McBSP0_RCR = 0x10040;    /* set rx control reg.   */
                                  /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_XCR = 0x10040;    /* set tx control reg.   */
                                  /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_SPCR = 0xC1A001;  /* setup SP control reg. */
  *(unsigned volatile int *)McBSP0_DXR = 0xAAAA;

  temp1 = *(unsigned volatile int *)McBSP0_DRR;
  temp1 = mcbsp0_read();
  mcbsp0_write(0x5555);
  temp = mcbsp0_read();
  temp &= 0xffff;

  if (temp != 0x5555)
  {
    temp = 1;
    return temp;
  }
```

```
  mcbsp0_write(0xaaaa);
  temp = mcbsp0_read();
  temp &= 0xffff;

  if (temp != 0xaaaa)
  {
    temp = 1;
    return temp;
  }

  return 0;
}

/*----------------------------------------------------------------------*/
/* mcbsp1_write() - used for write to McBSP1                            */
/*----------------------------------------------------------------------*/
void mcbsp0_write(int out_data)
{
  int temp;

  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;

  while ( temp == 0)  //POLE XRDY until non-zero
  {
    temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
  }

  *(unsigned volatile int *)McBSP0_DXR = out_data;
}

/*----------------------------------------------------------------------*/
/* mcbsp1_read() - used for read from McBSP1                            */
/*----------------------------------------------------------------------*/
int mcbsp0_read()
{
  int temp;

  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;

  while ( temp == 0) //POLE RRDY until non-zero
  {
    temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
  }
  temp = *(unsigned volatile int *)McBSP0_DRR;

  return temp;
}




/*----------------------------------------------------------------------*/
/*----------------------------------------------------------------------*/
/* mcbsp1_init() - used to initialize McBSP1                            */
/*----------------------------------------------------------------------*/
void mcbsp1_init()
{
        volatile unsigned int temp =0;
  /* set up McBSP0                                                      */
  *(unsigned volatile int *)McBSP1_SPCR = 0;      /* reset serial port     */
  *(unsigned volatile int *)McBSP1_PCR = 0;       /* set pin control reg.  */
  *(unsigned volatile int *)McBSP1_RCR = 0x10040; /* set rx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_XCR = 0x10040; /* set tx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_DXR = 0;
  *(unsigned volatile int *)McBSP1_SPCR = 0x12001;/*  setup SP control reg */
```

```
  printf("in McBSP1_init\n");
}


/*---------------------------------------------------------------------*/
/* mcbsp1_test() - used to test McBSP1                                 */
/*---------------------------------------------------------------------*/
int mcbsp1_test()
{


        volatile unsigned int temp =0,temp1;
  /* set up McBSP1 */
  *(unsigned volatile int *)McBSP1_SPCR = 0;        /* reset serial port     */
  *(unsigned volatile int *)McBSP1_SRGR = McBSP_BCKIN;/*set baud rate to 1.6276170*/
  *(unsigned volatile int *)McBSP1_PCR = 0xF00;      /* 0xA00set pin control reg.  */
  *(unsigned volatile int *)McBSP1_RCR = 0x10040;    /* set rx control reg.    */
                                       /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_XCR = 0x10040;    /* set tx control reg.    */
                                       /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP1_SPCR = 0xC1A001;  /* setup SP control reg. */
  *(unsigned volatile int *)McBSP1_DXR = 0xAAAA;

  temp1 = *(unsigned volatile int *)McBSP1_DRR;
  temp1 = mcbsp1_read();
  mcbsp1_write(0x5555);
  temp = mcbsp1_read();
  temp &= 0xffff;

  if (temp != 0x5555)
  {
    temp = 1;
    return temp;
  }

  mcbsp1_write(0xaaaa);
  temp = mcbsp1_read();
  temp &= 0xffff;

  if (temp != 0xaaaa)
  {
    temp = 1;
    return temp;
  }
  printf("in McBSP1_test\n");

  return 0;
}

/*---------------------------------------------------------------------*/
/* mcbsp1_write() - used for write to McBSP1                           */
/*---------------------------------------------------------------------*/
void mcbsp1_write(int out_data)
{
  int temp;

  temp = *(unsigned volatile int *)McBSP1_SPCR & 0x20000;

  while ( temp == 0)
  {
    temp = *(unsigned volatile int *)McBSP1_SPCR & 0x20000;
  }

  *(unsigned volatile int *)McBSP1_DXR = out_data;
}

/*---------------------------------------------------------------------*/
/* mcbsp1_read() - used for read from McBSP1                           */
/*---------------------------------------------------------------------*/
```

```
int mcbsp1_read()
{
  int temp;

  temp = *(unsigned volatile int *)McBSP1_SPCR & 0x2;
  /*Check if Receiver is ready with data to be read from DRR*/

  while ( temp == 0) /*stay until receiver is ready with data to be read from DRR*/
  {
    temp = *(unsigned volatile int *)McBSP1_SPCR & 0x2;
  }
  temp = *(unsigned volatile int *)McBSP1_DRR;/*read data from DRR*/

  return temp;
}

/*---------------------------------------------------------------------------*/


/*---------------------------------------------------------------------------*/
/* get_ioport() - used to read state of ioport                               */
/*---------------------------------------------------------------------------*/
unsigned int get_ioport(void)
{
  return(*(unsigned volatile int *)IO_PORT);
}

/*---------------------------------------------------------------------------*/
/* get_bdrev() - used to read board revision bits                            */
/*---------------------------------------------------------------------------*/
unsigned int get_bdrev(void)
{
  return(((*(unsigned volatile int *)IO_PORT)>>29)&0x7);
}

/*---------------------------------------------------------------------------*/
/* delay_msec() - used to delay DSP by user specified time in msec           */
/*---------------------------------------------------------------------------*/
void delay_msec(short msec)
{
/* assume 150 MHz CPU timer period = 4/150 MHz */
  int timer_limit = (msec*9375)<<2;
  int time_start;

  timer0_start();
  time_start = timer0_read();
  while ((timer0_read()-time_start) < timer_limit);

}

/*---------------------------------------------------------------------------*/
/* timer0_read() - used to read TIMER0 count                                 */
/*---------------------------------------------------------------------------*/
int timer0_read()
{
        int i;
        i = *(unsigned volatile int *)TIMER0_COUNT;
        return i;
}

/*---------------------------------------------------------------------------*/
/* timer0_start() - used to start TIMER0                                     */
/*---------------------------------------------------------------------------*/
void timer0_start()
{

        *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;    /* hold the timer    */
        *(unsigned volatile int *)TIMER0_CTRL |= 0x200;     /* use CPU CLK/4     */
        *(unsigned volatile int *)TIMER0_PRD  |= 0x30D4;          /*set for 32 bit cnt */
```

```
        *(unsigned volatile int *)TIMER0_CTRL |= 0xC0;      /* start the timer   */
        printf("in timer0_start\n");
}

/*---------------------------------------------------------------------------*/
/* timer0_init() - used to initialize TIMER0                                 */
/*---------------------------------------------------------------------------*/
void timer0_init()
{
        *(unsigned volatile int *)TIMER0_CTRL &= 0xFC2A;/* previously 0xfe2ahold the timer       */
        *(unsigned volatile int *)TIMER0_CTRL |= 0x000; /*0x200 for CPU CLK/4 or 000 for external clock source*/
        *(unsigned volatile int *)TIMER0_PRD   = 0x10; /* 0x10for 96kHZ out*/
        *(unsigned volatile int *)TIMER0_CTRL |= 0xC0; /* previously 0x3C0 start the timer      */
   printf("in timer0_init\n");                  /* enable timer0 int     */
}

/*---------------------------------------------------------------------------*/
/* timer0_isr() - interrupt service routine for TIMER0                       */
/* this routine also affects TOUT1.                                          */
/* External clock @ 13.043624MHz = orange                                    */
/* TOUT0 = lrcin = 48kHz with McBSP data clock to tinp0  CODEC               */
/*NOTE:  in this version sclk is from function generator                     */
/*---------------------------------------------------------------------------*/
interrupt void timer0_isr()
{

  int iData;
  *(unsigned volatile int *)CIPR = 0xffff;   /* clear all pending edma interrupts */
  *(unsigned volatile int *)CIPR |= 0x4000; /*previously 0x4000*/

  *(unsigned volatile int *)TIMER0_CTRL ^=0x4; //toggle datout
  iData = *(unsigned volatile int *)McBSP1_DRR;
  *(unsigned volatile int *)McBSP1_DXR =  iData;//stream data from DRR to DXR


}

/*---------------------------------------------------------------------------*/
/* timer0_init() - used to initialize TIMER0                                 */
/*---------------------------------------------------------------------------*/
void timer1_init()
{
        *(unsigned volatile int *)TIMER1_CTRL &= 0xFC2A;/* previously 0xfe2ahold the timer       */
        *(unsigned volatile int *)TIMER1_CTRL |= 0x000; /*0x200 for CPU CLK/4 or 000 for external clock source*/
        *(unsigned volatile int *)TIMER1_PRD   = 0x10; /* 0x10for 96kHZ out*/
        *(unsigned volatile int *)TIMER1_CTRL |= 0xC0; /* previously 0x3C0 start the timer      */
              /* enable timer0 int     */
}



/*
Function: void edma_init(void)
Description:
Initializes Enhanced DMA  registers and sets up address' to move data.

Pseudo Code:
1. Clear all pending EDMA events
2. Enable Events being used
3. Clear all pending EDMA interrupts
4. Set up EDMA for mcbsp1 transmitter
5. Set up EDMA for mcbsp1 receiver
6. Set up EDMA reload parameters for mcbsp1 transmitter
7. Set up EDMA reload parameters for mcbsp1 receiver
8. EXIT


*/
```

```
void edma_init()
{
  *(unsigned volatile int *)ECR = 0xffff;    /* clear all pending edma event */
  *(unsigned volatile int *)EER = 0x3000;    /* enable event 12 & 13 */

  *(unsigned volatile int *)CIPR = 0xffff;   /* clear all pending edma interrupts */
  *(unsigned volatile int *)CIER = 0x100;    /* previously 0x100 enable CCE8 */

  *(unsigned volatile int *)(EVENTC_PARAMS+OPT) = 0x49000002;
  *(unsigned volatile int *)(EVENTC_PARAMS+SRC) = McBSP1_DRR;
  *(unsigned volatile int *)(EVENTC_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTC_PARAMS+DST) = McBSP1_DXR;
  *(unsigned volatile int *)(EVENTC_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTC_PARAMS+LNK) = EVENTN_PARAMS;
    // set up edma reload params for mcbsp0 transmitter
  *(unsigned volatile int *)(EVENTN_PARAMS+OPT) = 0x49000002;
  *(unsigned volatile int *)(EVENTN_PARAMS+SRC) = McBSP1_DRR;
  *(unsigned volatile int *)(EVENTN_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTN_PARAMS+DST) = McBSP1_DXR;
  *(unsigned volatile int *)(EVENTN_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTN_PARAMS+LNK) = EVENTN_PARAMS;

/*  pcm_in = (int)&in;
  pcm_out = (int)&out;
  in_ptr = (short *)pcm_in;
  out_ptr = (short *)( pcm_out + DELAY*BLOCK_SIZE*DATA_SIZE);

  dma_index = 1;
  // set up edma for mcbsp0 transmitter
  *(unsigned volatile int *)(EVENTC_PARAMS+OPT) = 0x49000002;
  *(unsigned volatile int *)(EVENTC_PARAMS+SRC) = pcm_out;
  *(unsigned volatile int *)(EVENTC_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTC_PARAMS+DST) = McBSP1_DXR;
  *(unsigned volatile int *)(EVENTC_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTC_PARAMS+LNK) = EVENTN_PARAMS;

  // set up edma for mcbsp0 receiver
  *(unsigned volatile int *)(EVENTD_PARAMS+OPT) = 0x48380002;
  *(unsigned volatile int *)(EVENTD_PARAMS+SRC) = McBSP1_DRR;
  *(unsigned volatile int *)(EVENTD_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTD_PARAMS+DST) = pcm_in;
  *(unsigned volatile int *)(EVENTD_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTD_PARAMS+LNK) = EVENTO_PARAMS;

  src = pcm_out+BLOCK_SIZE*DATA_SIZE;
  dst = pcm_in+BLOCK_SIZE*DATA_SIZE;

  // set up edma reload params for mcbsp0 transmitter
  *(unsigned volatile int *)(EVENTN_PARAMS+OPT) = 0x49000002;
  *(unsigned volatile int *)(EVENTN_PARAMS+SRC) = src;
  *(unsigned volatile int *)(EVENTN_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTN_PARAMS+DST) = McBSP1_DXR;
  *(unsigned volatile int *)(EVENTN_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTN_PARAMS+LNK) = EVENTN_PARAMS;

  // set up edma reload params for mcbsp0 receiver
  *(unsigned volatile int *)(EVENTO_PARAMS+OPT) = 0x48380002;
  *(unsigned volatile int *)(EVENTO_PARAMS+SRC) = McBSP1_DRR;
  *(unsigned volatile int *)(EVENTO_PARAMS+CNT) = BLOCK_SIZE;
  *(unsigned volatile int *)(EVENTO_PARAMS+DST) = dst;
  *(unsigned volatile int *)(EVENTO_PARAMS+IDX) = 0;
  *(unsigned volatile int *)(EVENTO_PARAMS+LNK) = EVENTO_PARAMS;
  */
   printf("In EMDA_INIT\n");
}

interrupt void edma_isr()
{
    int i,temp;
```

```
    printf("Edma_isr\n");

    if ( dma_index  == NUM_OF_BLOCKS -1)
    {
                    dma_index =0;
                    dst = pcm_in;
                    src =  pcm_out;
    }
    else
    {
                    dma_index++;
                    src += BLOCK_SIZE*DATA_SIZE;
                    dst += BLOCK_SIZE*DATA_SIZE;
    }

    *(unsigned volatile int *)CIPR |= 0x100;  /* prevously 0x100 clear CCE8 bit  */
    *(unsigned volatile int *)(EVENTN_PARAMS+SRC) = src;
    *(unsigned volatile int *)(EVENTO_PARAMS+DST) = dst;

    temp = (int)in_ptr - pcm_in - NUM_OF_BLOCKS*BLOCK_SIZE*DATA_SIZE;
    if (temp >= 0)
    {
      in_ptr = (short *)pcm_in;

    }

    temp = (int)out_ptr - pcm_out - NUM_OF_BLOCKS*BLOCK_SIZE*DATA_SIZE;
    if (temp >= 0)
    {
      out_ptr = (short *)pcm_out;

    }

    for (i=0;i<BLOCK_SIZE;i++) *out_ptr++ = *in_ptr++ & 0xfffe;
}



/*****************************************************************************
 * NAME:        Device_Selection_Protocol ()                                 *
 * Author:  Lucius W. Perreault                                              *
 * Function: Ensures that the Hard drive is not busy (See page 223)          *
 * Psuedo code:                                                              *
 *        1)  Read status or alternate status regs                           *
 *        2)        LOOP back to 1 until BSY and DRQ = 0 of status reg       *
 *        3)        Write the Device/Head register with DEV = 0              *
 *        4)  Delay > 400 ns                                                 *
 *        5)  Read Status or alternate Regs until BSY and DRQ = 0            *
 *        6)  Exit                                                           *
 *****************************************************************************
*/
void Device_Selection_Protocol (void)
{
 unsigned int iStatus;
 iStatus = Read_Register(CB_STAT);

 while (((iStatus & CB_STAT_BSY) != 0) && ((iStatus & CB_STAT_DRQ) != 0))
  iStatus = Read_Register(CB_STAT); /*wait for BSY and DRQ = 0*/
 Write_Register(CB_DH,CB_DH_DEV0);/*set DEV for Master Device*/
 /*insert 400ns delay here*/
 while (((iStatus & CB_STAT_BSY) != 0) && ((iStatus & CB_STAT_DRQ) != 0))
  iStatus = Read_Register(CB_STAT); /*wait for BSY and DRQ = 0*/
}

/*****************************************************************************
 * NAME:        Write_Command_Parameters(cReg,iParam)
        *
 * Author:  Lucius W. Perreault                                              *
 * Function: Overwrites existing value of cReg with iParam                                  *
```

```
 ****************************************************************************
*/
int Write_Command_Parameters(volatile unsigned char * cReg,unsigned int iParam)
{
 unsigned int iStatus;
 Write_Register(cReg,iParam); /*write to specified register*/
  /*insert 400ns delay*/
 do
 iStatus = Read_Register(CB_STAT);
 while (((iStatus & CB_STAT_BSY) != 0) && ((iStatus & CB_STAT_DRQ) != 0));
 return(iStatus);     /*returns new value of cReg*/
}

/****************************************************************************
 * NAME:          Write_Command_Code(iParam)
          *
 * Author:  Lucius W. Perreault                                          *
 * Function: As the name implies it writes a command code to the CMD regs       *
 ****************************************************************************
*/
int Write_Command_Code(unsigned int iParam)
{
 unsigned int iStatus;

 Write_Register(CB_CMD,iParam);/*write command specified by iParam*/
 /*insert 400ns delay*/
 do
 iStatus = Read_Register(CB_STAT);
 while (((iStatus & CB_STAT_BSY) != 0) && ((iStatus & CB_STAT_BSY) != 0));
 return(iStatus);     /*returns new value of iStatus*/
}


/*********************************************************************
 * Name -  Read_Register(volatile unsigned char * cReg)            *
 * TYPE - Register to Register                                                                                    *
 * Author - Lucius W.  Perreault                                  *
 * Description: (see p.262)                                      *
 * Reads a byte from register specified in parameters              *
 * Psuedo Code:                                                  *
 *         1)        Set up address on A[6:2]                              *
 *         2)        Assert DIOR(pin 23 on HD) DB_ARE (pin 73 on J1)            *
 *         3)        Read Data on lower bits of data lines                    *
 *         4)  Negate DIOR(pin 23 on HD) DB_ARE (pin 73 on J1)            *
 *         5)  Return Data                                       *
 *********************************************************************/

int  Read_Register(volatile unsigned char * cReg)
{
          int uiData1;
          static int iCounter=1;
          static int iCounter1 = 1;
    uiData1 = uiData1;
    iCounter = iCounter;
    iCounter1 = iCounter1;

          (*(unsigned volatile int*)(CE2_SPACE)) = 0x0F;/*set up address*/
          /*Upper = DATA Lower=Address (DSP view)*/
          for (iCounter = 1; iCounter <= 0xFFFFF; iCounter ++);/*DELAY*/
          (*(unsigned volatile int*)(CE2_SPACE)) = 0x0F | DIOR;/*Set DIOR*/
          for (iCounter = 1; iCounter <= 0xFFFFF; iCounter ++);/*DELAY*/
          uiData1 = (*(unsigned volatile int*)(CE2_SPACE))&0xFFFF0000; /*read word*/
          for (iCounter = 1; iCounter <= 0xFFFFF; iCounter ++);/*DELAY*/
          (*(unsigned volatile int*)(CE2_SPACE)) &=  0xFFFF7FFF;/*clear DIOR*/
          for (iCounter = 1; iCounter <= 0xFFFFF; iCounter ++);/*DELAY*/
          printf("uiData1 = %d\n",iCounter);           /*  data/address*/

          /*lower word*/
          return(1);
```

```
}
/*
        int uiData1;
        static int iCounter=1;
        static int iCounter1 = 1;

   printf("uiData1 = %d\n",iCounter);
        (*(unsigned volatile int*)(CE2_SPACE)) = 0x1111000F | DIOR;
        //Upper = DATA Lower=Address (DSP view)
        uiData1 = (*(unsigned volatile int*)(CE2_SPACE))&0xFFFF0000; //read word

        printf("Data = %d\n",uiData1);
   iCounter=iCounter<<1;
   iCounter1++;
        if (iCounter1 == 33)
        {
        iCounter  = 1;
   iCounter1 = 1;
   }
*/
/*********************************************************************
 * Name -  Write_Register(volatile unsigned char * cReg)          *
 * TYPE -  Register to Register                                                    *
 * Author - Lucius W.  Perreault                                  *
 * Description: (see p.262)                                       *
 * Writes specified a byte to a register specified in parameters    *
 * Psuedo Code:                                                   *
 *        1)      Set up address on A[6:2]                        *
 *        2)      Assert DIOW(pin 23 on HD) DB_AWE (pin 74 on J1)         *
 *        3)      Read Data on lower bits of data lines                   *
 *        4)  Negate DIOW(pin 23 on HD) DB_AWE (pin 74 on J1)           *
 *        5)  Return Data                                         *
 *********************************************************************/

int  Write_Register(volatile unsigned char * cReg,unsigned int iParam)
{


  return(*cReg);
}

/*********************************************************************
 * Name -  DMA_Command_Protocol(unsigned int iCommand)                            *
 * TYPE -
        *
 * Author - Lucius W.  Perreault                                  *
 * Description: (see p.233)                                       *
 * Excecutes Code passed by iParam                                                 *
 * Psuedo Code:                                                   *
 *        1)      Execute Device_Selection Protocol                                *
 *        2)      Write required command parameters (see Table e.3 for regs)     *
 *        3)      Write Command Code to the Command Reg (CB_CMD)                        *
 *        4)      Check CB_STAT_ERR bit of CB_STAT for error if yes goto 9     *
 *        5)      Assert DMACK after DMARQ is asserted by hard drive (auto)    *
 *        6)      Transfer data as described in MultiWord DMA timing                  *
 *        7)      Is transfer complete? goto next line:goto #5 again                  *
 *        8)      Check CB_STAT_ERR bit of CB_STAT for error if no goto               *
 *        9)      Hard drive does some stuff                                                      *
 *        10)     Check for interrupt
        *
 *        11)     Read Status Reg (CB_STAT) to clear pending interrupt           *
 *********************************************************************/

int  DMA_Command_Protocol(unsigned int iCommand)
{

  Device_Selection_Protocol ();/*Run Device Selection Protocol*/

  switch(iCommand)/*Determine which Command Parameters to write to*/
```

```
{
        case (CMD_READ_DMA_QUEUED):
        case (CMD_WRITE_DMA_QUEUED):
                {
                Write_Command_Parameters(CB_FR,sRegister.Feature_Reg);
                Write_Command_Parameters(CB_SC,sRegister.Sector_Count);
                Write_Command_Parameters(CB_SN,sRegister.Sector_Number);
                Write_Command_Parameters(CB_CL,sRegister.Cylinder_Low);
                Write_Command_Parameters(CB_CH,sRegister.Cylinder_High);
                Write_Command_Parameters(CB_DH,sRegister.Device_Head);
                break;
                }
        case (CMD_CFA_ERASE_SECTORS):
        case (CMD_FLUSH_CACHE):
        case (CMD_READ_DMA):
        case (CMD_SET_MAX_ADDRESS):
        case (CMD_WRITE_DMA):
                {
                 Write_Command_Parameters(CB_SC,sRegister.Sector_Count);
                 Write_Command_Parameters(CB_SN,sRegister.Sector_Number);
                 Write_Command_Parameters(CB_CL,sRegister.Cylinder_Low);
                 Write_Command_Parameters(CB_CH,sRegister.Cylinder_High);
                 Write_Command_Parameters(CB_DH,sRegister.Device_Head);
                 break;
                }
        case (CMD_SEEK):
                {
                 Write_Command_Parameters(CB_SN,sRegister.Sector_Number);
                 Write_Command_Parameters(CB_CL,sRegister.Cylinder_Low);
                 Write_Command_Parameters(CB_CH,sRegister.Cylinder_High);
                 Write_Command_Parameters(CB_DH,sRegister.Device_Head);
                 break;
                }
  case (CMD_CFA_REQUEST_EXT_ERR_CODE):
  case (CMD_GET_MEDIA_STATUS):
  case (CMD_IDENTIFY_DEVICE):
  case (CMD_NOP):
        {
        Write_Command_Parameters(CB_DH,sRegister.Device_Head);
        break;
        }

 } /*end of case statement*/

 return(iCommand);
}
```

```
/*lwpdefine.h

        defines all my funtions.

*/
#define Bytes_Per_Sector      0x200   /*512*/
#define Bytes_Per_Cluster   0x4000    /*16384*/

/*drive specific defines based on 540 MBytes*/
#define Total_Clusters          0x80A9
#define Total_Sectors           0x101520

struct /*same as structure accept that you can only access one variable at a time*/
{

unsigned int    Data_Reg,
                                Data_Port_Reg,
                                Error_Reg,
                                Feature_Reg,
                                Sector_Count,
                                Sector_Number,
                                Cylinder_Low,
                                Cylinder_High,
                                Device_Head,
                                Primary_Status,
                                Command_Reg,
                                Alternate_Status_Reg,
                                Device_Control;
}sRegister;


void Device_Selection_Protocol (void);
int  DMA_Command_Protocol(unsigned int iCommand);

int  Read_Register(volatile unsigned char * cReg);
int  Write_Register(volatile unsigned char * cReg,unsigned int iParam);
int Write_Command_Parameters(volatile unsigned char * cReg,unsigned int iParam);
```

APPENDIX I

Hard Drive Code Listing

```
//hdcommands.h

// registers NOTE: These need an offset of somekind(CE2_OFFSET)
//Assume Data lines are A[6:2] (DSP) connect to
//          CS1     CS0     DA2     DA1     DA0   of the harddrive!!

#define CE2_OFFSET 0xA00000000
#define CE3_OFFSET 0xB00000000

#define CB_DATA   (volatile unsigned char *)0x08    /* data reg         in/out   2*/
#define CB_DATA_P (volatile unsigned char *)0x00    /* data Port reg    in/out   2*/
#define CB_ERR    (volatile unsigned char *)0x0B    /* error            in               1*/
#define CB_FR     (volatile unsigned char *)0x0B    /* feature reg         out           1*/
#define CB_SC     (volatile unsigned char *)0x0A    /* sector count     in/out          1*/
#define CB_SN     (volatile unsigned char *)0x0B    /* sector number    in/out          1*/
#define CB_CL     (volatile unsigned char *)0x0C    /* cylinder low     in/out          1*/
#define CB_CH     (volatile unsigned char *)0x0D    /* cylinder high    in/out          1*/
#define CB_DH     (volatile unsigned char *)0x0E    /* device head      in/out          1*/
#define CB_STAT   (volatile unsigned char *)0x0F    /* primary status   in              1*/
#define CB_CMD    (volatile unsigned char *)0x08    /* command             out          1*/
#define CB_ASTAT  (volatile unsigned char *)0x16    /* alternate status in                  1*/
#define CB_DC     (volatile unsigned char *)0x16    /* device control      out          1*/


// error reg (CB_ERR) bits

#define CB_ER_ICRC 0x80    // ATA Ultra DMA bad CRC
#define CB_ER_BBK  0x80    // ATA bad block
#define CB_ER_UNC  0x40    // ATA uncorrected error
#define CB_ER_MC   0x20    // ATA media change
#define CB_ER_IDNF 0x10    // ATA id not found
#define CB_ER_MCR  0x08    // ATA media change request
#define CB_ER_ABRT 0x04    // ATA command aborted
#define CB_ER_NTK0 0x02    // ATA track 0 not found
#define CB_ER_NDAM 0x01    // ATA address mark not found

#define CB_ER_P_SNSKEY 0xf0   // ATAPI sense key (mask)
#define CB_ER_P_MCR    0x08   // ATAPI Media Change Request
#define CB_ER_P_ABRT   0x04   // ATAPI command abort
#define CB_ER_P_EOM    0x02   // ATAPI End of Media
#define CB_ER_P_ILI    0x01   // ATAPI Illegal Length Indication

// bits 7-4 of the device/head (CB_DH) reg

#define CB_DH_DEV0 0xa0    // select device 0
#define CB_DH_DEV1 0xb0    // select device 1

// status reg (CB_STAT and CB_ASTAT) bits

#define CB_STAT_BSY  0x80  // busy
#define CB_STAT_RDY  0x40  // ready
#define CB_STAT_DF   0x20  // device fault
#define CB_STAT_WFT  0x20  // write fault (old name)
#define CB_STAT_SKC  0x10  // seek complete
#define CB_STAT_SERV 0x10  // service
#define CB_STAT_DRQ  0x08  // data request
#define CB_STAT_CORR 0x04  // corrected
#define CB_STAT_IDX  0x02  // index
#define CB_STAT_ERR  0x01  // error (ATA)
#define CB_STAT_CHK  0x01  // check (ATAPI)

// device control reg (CB_DC) bits

#define CB_DC_HD15   0x08  // bit should always be set to one
```

```
#define CB_DC_SRST   0x04  // soft reset
#define CB_DC_NIEN   0x02  // disable interrupts

//Lucius' Hard drive Controls

#define DIOR                0x8000



// Most mandtory and optional ATA commands (from ATA-3),

#define CMD_CFA_ERASE_SECTORS           0xC0
#define CMD_CFA_REQUEST_EXT_ERR_CODE    0x03
#define CMD_CFA_TRANSLATE_SECTOR        0x87
#define CMD_CFA_WRITE_MULTIPLE_WO_ERASE 0xCD
#define CMD_CFA_WRITE_SECTORS_WO_ERASE  0x38
#define CMD_CHECK_POWER_MODE1           0xE5
#define CMD_CHECK_POWER_MODE2           0x98
#define CMD_DEVICE_RESET                0x08
#define CMD_EXECUTE_DEVICE_DIAGNOSTIC   0x90
#define CMD_FLUSH_CACHE                 0xE7
#define CMD_FORMAT_TRACK                0x50
#define CMD_IDENTIFY_DEVICE             0xEC
#define CMD_IDENTIFY_DEVICE_PACKET      0xA1
#define CMD_IDENTIFY_PACKET_DEVICE      0xA1
#define CMD_IDLE1                       0xE3
#define CMD_IDLE2                       0x97
#define CMD_IDLE_IMMEDIATE1             0xE1
#define CMD_IDLE_IMMEDIATE2             0x95
#define CMD_INITIALIZE_DRIVE_PARAMETERS 0x91
#define CMD_INITIALIZE_DEVICE_PARAMETERS 0x91
#define CMD_NOP                         0x00
#define CMD_PACKET                      0xA0
#define CMD_READ_BUFFER                 0xE4
#define CMD_READ_DMA                    0xC8
#define CMD_READ_DMA_QUEUED             0xC7
#define CMD_READ_MULTIPLE               0xC4
#define CMD_READ_SECTORS                0x20
#define CMD_READ_VERIFY_SECTORS         0x40
#define CMD_RECALIBRATE                 0x10
#define CMD_SEEK                        0x70
#define CMD_SET_FEATURES                0xEF
#define CMD_SET_MULTIPLE_MODE           0xC6
#define CMD_SLEEP1                      0xE6
#define CMD_SLEEP2                      0x99
#define CMD_STANDBY1                    0xE2
#define CMD_STANDBY2                    0x96
#define CMD_STANDBY_IMMEDIATE1          0xE0
#define CMD_STANDBY_IMMEDIATE2          0x94
#define CMD_WRITE_BUFFER                0xE8
#define CMD_WRITE_DMA                   0xCA
#define CMD_WRITE_DMA_QUEUED            0xCC
#define CMD_WRITE_MULTIPLE              0xC5
#define CMD_WRITE_SECTORS               0x30
#define CMD_WRITE_VERIFY                0x3C
#define CMD_SET_MAX_ADDRESS                                     0xF9
#define CMD_GET_MEDIA_STATUS                            0xDA
```

```
/*-------------------------------------------------------------------*/
/* FILENAME: dsk6xHSB.h - HandShakingBuffer Header File              */
/*                                                                   */
/*  Rev 2.0   17 July 2000  T.J.Dillon                               */
/*                                                                   */
/*-------------------------------------------------------------------*/

#define HS_BUFFER_LEN                          6
#define ID_0                                                                0x12345678
#define ID_1                                                                0xf1f2f3f4
#define ID_2                                                                0x08070605
// 4th is the command
#define ISRAM                                  0x1000
#define SDRAM                                  0x2000
#define FLASH                                  0x3000
#define MCBSP                                  0x4000
#define TIMER                                  0x5000
#define QDMA                                   0x6000
#define LEDS                                   0x7000
#define CODEC 0x8000
#define MENU  0x9000
#define PGMEM 0xA000
// 5th is the status register
#define HOST_RECEIVE_HAND_SHAKING_INFO     0x55555555
#define HOST_STATUS_INPUT_READY                                             0x77777777
#define HOST_STATUS_END_PROCESSING                                  0xFFFFFFFF
#define DSP_STATUS_OUTPUT_READY                                             0x66666666
#define DSP_PROCESSING_COMMAND                                              0x88888888
// 6th is return error code
#define ISRAM_OK                               0x10000000
#define ISRAM_ERR55 0x10005555
#define ISRAM_ERRAA 0x1000AAAA
#define ISRAM_ERROR 0x1000FFFF
#define SDRAM_OK                               0x20000000
#define SDRAM_ERR55 0x20005555
#define SDRAM_ERRAA 0x2000AAAA
#define SDRAM_ERR5A 0x20005A5A
#define SDRAM_ERRA5 0x2000A5A5
#define SDRAM_ERROR 0x2000FFFF
#define FLASH_OK                               0x30000000
#define FLASH_ERRCK 0x30003333
#define FLASH_ERR55 0x30005555
#define FLASH_ERR77 0x30007777
#define FLASH_ERRAA 0x3000AAAA
#define FLASH_ERRBB 0x3000BBBB
#define FLASH_ERROR 0x3000FFFF
#define MCBSP_OK                               0x40000000
#define MCBSP_ERROR 0x4000FFFF
#define TIMER_OK                               0x50000000
#define TIMER_ERROR 0x5000FFFF
#define QDMA_OK     0x60000000
#define QDMA_ERROR  0x6000FFFF
#define LEDS_OK     0x70000000
#define LEDS_ERROR  0x7000FFFF
#define CODEC_OK    0x80000000
#define CODEC_ERRTN 0x80005555
#define CODEC_ERRCD 0x8000AAAA
#define CODEC_ERROR 0x8000FFFF
#define PGMEM_OK    0xA0000000
#define PGMEM_ERROR 0xA000FFFF
#define TEST_DISABLED 0xFF00FFFF
#define DEFAULT_ERROR 0xF000FFFF
```

```
/*-----------------------------------------------------------------------*/
/* FILENAME: cnfdsp.h - Confidence Test Header File                      */
/*                                                                       */
/*  Rev 2.0   17 July 2000  T.J.Dillon                                   */
/*                                                                       */
/*-----------------------------------------------------------------------*/

#define  MEMTST 1000
#define  FLATST 2000
#define  ENDPNT 3000
#define  INTERV 4000
#define  PRINT 1
#define  SPECIAL MEMTST

int  mem_test (int pattern, int start_adress, int size_in_word );
int  mem_test_alt (int pattern, int start_adress, int size_in_word );
//int  flash_checksum (int start_address, int size_in_byte);
int  flash_test ( char pattern, int start_address, int size_in_byte );
int  flash_page_prog( char pattern, char *start_address, int page_size);
int  play_codec(int number, int tone_playbk);
int  mcbsp0_test();
void mcbsp0_init();
int  mcbsp0_read();
void mcbsp0_write(int out_data);

//-------------------------------------------------------------------------
int  mcbsp1_test();
void mcbsp1_init();
int  mcbsp1_read();
void mcbsp1_write(int out_data);
void edma_init();
void emif_init();

//-------------------------------------------------------------------------
void qdma_start(int src, int dst, int size);
int  qdma_check(int src, int dst, int size);
void led_blink_all(int count, int ms_period);
void led_blink(int count, int ms_period, unsigned int leds_to_light);
unsigned int get_ioport(void);
unsigned int get_bdrev(void);
void delay_msec(short msec);
void timer1_init();
void timer0_init();
void timer0_start();
int  timer0_read();
```

```
/*****************************************************************************
* FILENAME
*   c6211dsk.h
*
* DESCRIPTION
*   DSK Header File
*
*****************************************************************************/

/* Register definitions for C6211 chip on DSK */

/* Define EMIF Registers  */
#define EMIF_GCR                                0x1800000           /* Address of EMIF global control          */
#define EMIF_CE0                                0x1800008           /* Address of EMIF CE0 control             */
#define EMIF_CE1                                0x1800004           /* Address of EMIF CE1 control             */
/*****************************************************************************/
#define EMIF_CE2                                0x1800010           /* Address of EMIF CE2 control             */
#define EMIF_CE3                                0x1800014           /* Address of EMIF CE3 control             */
/*****************************************************************************/

#define EMIF_SDCTRL                             0x1800018           /* Address of EMIF SDRAM control           */
#define EMIF_SDRP                               0x180001c           /* Address of EMIF SDRM refresh period     */
#define EMIF_SDEXT                              0x1800020           /* Address of EMIF SDRAM extension         */

/* Define McBSP0 Registers */
#define McBSP0_DRR      0x18c0000   /* Address of data receive reg.        */
#define McBSP0_DXR      0x18c0004   /* Address of data transmit reg.       */
#define McBSP0_SPCR     0x18c0008   /* Address of serial port contl. reg.  */
#define McBSP0_RCR      0x18c000C   /* Address of receive control reg.     */
#define McBSP0_XCR      0x18c0010   /* Address of transmit control reg.    */
#define McBSP0_SRGR     0x18c0014   /* Address of sample rate generator    */
#define McBSP0_MCR      0x18c0018   /* Address of multichannel reg.        */
#define McBSP0_RCER     0x18c001C   /* Address of receive channel enable.  */
#define McBSP0_XCER     0x18c0020   /* Address of transmit channel enable. */
#define McBSP0_PCR      0x18c0024   /* Address of pin control reg.         */

/* Define McBSP1 Registers */
#define McBSP1_DRR      0x1900000   /* Address of data receive reg.        */
#define McBSP1_DXR      0x1900004   /* Address of data transmit reg.       */
#define McBSP1_SPCR     0x1900008   /* Address of serial port contl. reg.  */
#define McBSP1_RCR      0x190000C   /* Address of receive control reg.     */
#define McBSP1_XCR      0x1900010   /* Address of transmit control reg.    */
#define McBSP1_SRGR     0x1900014   /* Address of sample rate generator    */
#define McBSP1_MCR      0x1900018   /* Address of multichannel reg.        */
#define McBSP1_RCER     0x190001C   /* Address of receive channel enable.  */
#define McBSP1_XCER     0x1900020   /* Address of transmit channel enable. */
#define McBSP1_PCR      0x1900024   /* Address of pin control reg.         */

/* Define L2 Cache Registers */
#define L2CFG           0x1840000   /* Address of L2 config reg            */
#define MAR0            0x1848200   /* Address of mem attribute reg        */

/* Define Interrupt Registers */
#define IMH             0x19c0000   /* Address of Interrupt Multiplexer High*/
#define IML             0x19c0004   /* Address of Interrupt Multiplexer Low */

/* Define Timer0 Registers */
#define TIMER0_CTRL     0x1940000     /* Address of timer0 control reg.     */
#define TIMER0_PRD      0x1940004     /* Address of timer0 period reg.      */
#define TIMER0_COUNT    0x1940008     /* Address of timer0 counter reg.     */

/* Define Timer1 Registers */
#define TIMER1_CTRL     0x1980000     /* Address of timer1 control reg.     */
#define TIMER1_PRD      0x1980004     /* Address of timer1 period reg.      */
#define TIMER1_COUNT    0x1980008     /* Address of timer1 counter reg.     */

/* Define EDMA Registers */
#define PQSR                                    0x01A0FFE0          /* Address of priority queue status    */
#define CIPR                                    0x01A0FFE4          /* Address of channel interrupt pending */
#define CIER                                    0x01A0FFE8          /* Address of channel interrupt enable  */
```

```
#define CCER                                0x01A0FFEC          /* Address of channel chain enable    */
#define ER                                  0x01A0FFF0          /* Address of event register          */
#define EER                                 0x01A0FFF4          /* Address of event enable register   */
#define ECR                                 0x01A0FFF8          /* Address of event clear register    */
#define ESR                                 0x01A0FFFC          /* Address of event set register      */

/* Define EDMA Transfer Parameter Entry Fields */
#define OPT                                 0*4                 /* Options Parameter                  */
#define SRC                                 1*4                 /* SRC Address Parameter              */
#define CNT                                 2*4                 /* Count Parameter                    */
#define DST                                 3*4                 /* DST Address Parameter              */
#define IDX                                 4*4                 /* IDX Parameter                      */
#define LNK                                 5*4                 /* LNK Parameter                      */

/* Define EDMA Parameter RAM Addresses */
#define EVENT0_PARAMS 0x01A00000
#define EVENT1_PARAMS EVENT0_PARAMS + 0x18
#define EVENT2_PARAMS EVENT1_PARAMS + 0x18
#define EVENT3_PARAMS EVENT2_PARAMS + 0x18
#define EVENT4_PARAMS EVENT3_PARAMS + 0x18
#define EVENT5_PARAMS EVENT4_PARAMS + 0x18
#define EVENT6_PARAMS EVENT5_PARAMS + 0x18
#define EVENT7_PARAMS EVENT6_PARAMS + 0x18
#define EVENT8_PARAMS EVENT7_PARAMS + 0x18
#define EVENT9_PARAMS EVENT8_PARAMS + 0x18
#define EVENTA_PARAMS EVENT9_PARAMS + 0x18
#define EVENTB_PARAMS EVENTA_PARAMS + 0x18
#define EVENTC_PARAMS EVENTB_PARAMS + 0x18
#define EVENTD_PARAMS EVENTC_PARAMS + 0x18
#define EVENTE_PARAMS EVENTD_PARAMS + 0x18
#define EVENTF_PARAMS EVENTE_PARAMS + 0x18
#define EVENTN_PARAMS EVENTF_PARAMS + 0x18
#define EVENTO_PARAMS EVENTN_PARAMS + 0x18

/* Define QDMA Memory Mapped Registers */
#define QDMA_OPT                            0x02000000          /* Address of QDMA options register   */
#define QDMA_SRC                            0x02000004          /* Address of QDMA SRC address register */
#define QDMA_CNT                            0x02000008          /* Address of QDMA counts register    */
#define QDMA_DST                            0x0200000C          /* Address of QDMA DST address register */
#define QDMA_IDX                            0x02000010          /* Address of QDMA index register     */

/* Define QDMA Pseudo Registers */
#define QDMA_S_OPT                          0x02000020          /* Address of QDMA options register   */
#define QDMA_S_SRC                          0x02000024          /* Address of QDMA SRC address register */
#define QDMA_S_CNT                          0x02000028          /* Address of QDMA counts register    */
#define QDMA_S_DST                          0x0200002C          /* Address of QDMA DST address register */
#define QDMA_S_IDX                          0x02000030          /* Address of QDMA index register     */

/* Definitions for the DSK Board and SW */
#define PI                                  3.1415926
#define IO_PORT                                         0x90080000  /* I/O port Address,top byte valid data */
#define INTERNAL_MEM_SIZE (0x4000)>>2
#define EXTERNAL_MEM_SIZE (0x400000)>>2
#define FLASH_SIZE                          0x20000
#define POST_SIZE                           0x10000
#define FLASH_WRITE_SIZE 0x80
#define INTERNAL_MEM_START 0xc000
#define EXTERNAL_MEM_START 0x80000000
#define FLASH_START                         0x90000000
#define POST_END                            0x90010000
#define FLASH_ADR1                          0x90005555
#define FLASH_ADR2                          0x90002AAA
#define FLASH_KEY1                          0xAA
#define FLASH_KEY2                          0x55
#define FLASH_KEY3                          0xA0
#define ALL_A                               0xaaaaaaaa
#define ALL_5                               0x55555555
#define CE1_8                   0xffffff03  /* reg to set CE1 as 8bit async */
/*****************************************************************/
#define CE1_16                  0xffffff13  /* reg to set CE1 as 8bit async */
```

```
/*********************************************************************/
#define CE1_32                  0xffffff23  /* reg to set CE1 as 32bit async */
#define McBSP_1M                                    0x2014004A  /* set baud rate to 1MHz */
#define McBSP_25M                                   0x20140002  /*set baud rate to 512(48K)*/
#define McBSP_18M                                   0x20140003  /*set baud rate to 384(48K)*/
#define McBSP_12M                                   0x20140005  /*set baud rate to 256(48K)*/
#define McBSP_BCKIN                                 0x2014002D  /*2014002Dset baud rate to 1.6276170*/

#define CE2_SPACE                                   0xA0000000  /*External Memory Space CE2*/
#define CE3_SPACE           0xB0000000  /*External Memory Space CE2*/
```

```
*-----------------------------------------------------------------------*/
/* FILENAME: cnfdsp.c -- DSK Confidence test - DSP Code                   */
/*                                                                       */
/*  Rev 2.17   21 Sep  2000  Z.Zhang, T.J.Dillon                         */
/*                                                                       */
/*-----------------------------------------------------------------------*/
/*  HISTORY                                                              */
/*  Rev 1.00  Created by Z.Zhang                                         */
/*  Rev 1.10  Mods a) LEDS and CODEC parameter passing                   */
/*                b) Removed FLASH checksum                              */
/*  Rev 1.20  Mods - Reorganized to be consistent with cnfDSP.c          */
/*  Rev 1.30  Mods - Added board revision info                           */
/*  Rev 2.00  Mods - Updated for C6711 DSK                               */
/*  Rev 2.10  Mods - Updated for Larger SDRAMs                           */
/*  Rev 2.16  Mods - Added newer memory test for SDRAM                   */
/*  Rev 2.17  Mods - Added -j option to turn off codec test (JP1 Installed)*/
/*                                                                       */
/*-----------------------------------------------------------------------*/

#include <stdio.h>
#include <c6x.h>
#include "c6211dsk.h"
#include "cnfDSP.h"
#include "dsk6xHSB.h"

#pragma DATA_SECTION(handShakingBuffer, "my_DataSect")
int handShakingBuffer[HS_BUFFER_LEN];

short sin_table[] = {0, 23170>>2, 32767>>2, 23170>>2,0, -23170>>2, -32767>>2, -23170>>2};
int flag;
//static int exp_chksum,cmp_chksum;

/*-----------------------------------------------------------------------*/
/* main()                                                                */
/*-----------------------------------------------------------------------*/

int main()
{
  int src,dst,size; /* Used for QDMA */
  int error,errortone;
  unsigned int io_port_values;
  unsigned int user_dip_settings;
  unsigned int pwb_assy_rev;
  int ext_mem_size;
  /* dsp and peripheral initialization */
  CSR=0x100;                                                                    /* disab
  IER=1;                        /* disable all interrupts except NMI */
  ICR=0xffff;                    /* clear all pending interrupts      */

/*************************************************************************
* Standard 6211 DSK includes 2 MT48LC1M16A1-7 devices =>4MB SDRAM        *
*   16Mb (16-bit x 2 banks x 512K) parts = 2MB / part                    *
*   EMIF_SDCTRL=0x07227000                                               *
*   EMIF_SDEXT=0x54529                    Board Rev = 1                  *
*----------------------------------------------------------------------*
* Standard 6711 DSK includes 2 MT48LC4M16A2-8 devices =>16MB SDRAM       *
*   64Mb (16-bit x 4 banks x 1M)   parts = 8MB / part                    *
*   EMIF_SDCTRL=0x57116000                                               *
*   EMIF_SDEXT=0x54529 (Hitachi 0x54509)      Board Rev = 2              *
*----------------------------------------------------------------------*
* Other 6711 DSK configurations are as follows:                          *
*   128Mb (16-bit x 4 banks x 2M)  parts = 16MB / part (=>32MB SDRAM)    *
*   EMIF_SDCTRL=0x53116000                                               *
*   EMIF_SDEXT=0x54529 (Hitachi 0x54509)      Board Rev = 3              *
*   ----------------------------------------------------------------------*
*   256Mb (16-bit x 4 banks x 4M)  parts = 32MB / part (=>64MB SDRAM)    *
*   EMIF_SDCTRL=0x63116000                                               *
*   EMIF_SDEXT=0x54529                    Board Rev = 4                  *
*************************************************************************/
```

```
  *(unsigned volatile int *)EMIF_GCR = 0x3300; /* EMIF global control   */
  *(unsigned volatile int *)EMIF_CE0 = 0x30;   /* EMIF CE0control       */
  *(unsigned volatile int *)EMIF_CE1 = 0xffffff03; /* EMIF CE1 control, 8bit async */
  *(unsigned volatile int *)EMIF_SDCTRL = 0x07227000; /* EMIF SDRAM control     */
  *(unsigned volatile int *)EMIF_SDRP = 0x61a;       /* EMIF SDRM refresh period */
  *(unsigned volatile int *)EMIF_SDEXT = 0x54529; /* EMIF SDRAM extension    */

#if PRINT
  printf("\n*TMS320C6211/6711 DSK Confidence Test*\n");
#endif

  /*-------------------------------------------------------------------------*/
  /* Read DIP switches                                                       */
  /*-------------------------------------------------------------------------*/
  *(unsigned volatile int *)EMIF_CE1 = CE1_32;// EMIF CE1 control, 32bit async
  io_port_values=get_ioport();
  user_dip_settings=io_port_values&0x7000000;
  pwb_assy_rev=get_bdrev();
  *(unsigned volatile int *)EMIF_CE1 = CE1_8;// EMIF CE1 control, 8bit async

#if PRINT
/*  printf("\n USER_SW3=%d ",(user_dip_settings&0x4000000)>>26);
  printf(" USER_SW2=%d ",(user_dip_settings&0x2000000)>>25);
  printf(" USER_SW1=%d ",(user_dip_settings&0x1000000)>>24);
*/
  printf("\n USER_SW1=%d ",(user_dip_settings&0x1000000) ? 0:1);
  printf("\n USER_SW2=%d ",(user_dip_settings&0x2000000) ? 0:1);
  printf("\n USER_SW3=%d ",(user_dip_settings&0x4000000) ? 0:1);
  printf(" Board Rev=%d\n\n",pwb_assy_rev);
#endif

  /*-------------------------------------------------------------------------*/
  /* Modify SDRAM parameter value(s) according to board revision           */
  /*-------------------------------------------------------------------------*/
  if(pwb_assy_rev ==1)
  {
    *(unsigned volatile int *)EMIF_SDCTRL = 0x07227000;
    ext_mem_size=EXTERNAL_MEM_SIZE;
  }
  else if(pwb_assy_rev ==2)
  {
    *(unsigned volatile int *)EMIF_SDCTRL = 0x57116000;
    ext_mem_size=4*EXTERNAL_MEM_SIZE;
  }
  else if(pwb_assy_rev ==3)
  {
    *(unsigned volatile int *)EMIF_SDCTRL = 0x53116000;
    ext_mem_size=8*EXTERNAL_MEM_SIZE;
  }
  else if(pwb_assy_rev ==4)
  {
    *(unsigned volatile int *)EMIF_SDCTRL = 0x63116000;
    ext_mem_size=16*EXTERNAL_MEM_SIZE;
  }
  else /* if(pwb_assy_rev ==0) */
  {
    *(unsigned volatile int *)EMIF_SDCTRL = 0x07126000;
    ext_mem_size=EXTERNAL_MEM_SIZE;
  }

  /*-------------------------------------------------------------------------*/
  /* Perform Hand Shaking                                                    */
  /*-------------------------------------------------------------------------*/
  handShakingBuffer[0] = ID_0;
  handShakingBuffer[1] = ID_1;
  handShakingBuffer[2] = ID_2;
  handShakingBuffer[3] = 0;
  handShakingBuffer[4] = 0;
  handShakingBuffer[5] = io_port_values;
#if PRINT
```

```
  printf("Starting to Handshake.\n");
#endif
  /*-------------------------------------------------------------------------*/
  /* Wait for Host ACK                                                       */
  /*-------------------------------------------------------------------------*/
  while(handShakingBuffer[4] != HOST_RECEIVE_HAND_SHAKING_INFO);
#if PRINT
  printf("look here.\n");
#endif
  /*-------------------------------------------------------------------------*/
  /* Reset handShakingBuffer                                                 */
  /*-------------------------------------------------------------------------*/
  handShakingBuffer[0] = 0;
  handShakingBuffer[1] = 0;
  handShakingBuffer[2] = 0;

#if PRINT
  printf("Hand Shake Complete.\n");
#endif

  /*-------------------------------------------------------------------------*/
  /* Run DSP forever                                                         */
  /*-------------------------------------------------------------------------*/
  while(1)
  {

  /*-------------------------------------------------------------------------*/
  /* Test for HOST ready                                                     */
  /*-------------------------------------------------------------------------*/
#if PRINT
    printf("\nDSP Ready for HOST Command.\n");
#endif

    /*-------------------------------------------------------------------------*/
    /* Wait until HOST has a command and parameters ready                      */
    /*-------------------------------------------------------------------------*/
                                  //  while(handShakingBuffer[4] != HOST_STATUS_INPUT_READY
                                                                  //  && handShakingBuffer[4] != HOST_STATUS_END_PROCESSING);

    /*-------------------------------------------------------------------------*/
    /* Quit if HOST requests it or Process a HOST command                      */
    /*-------------------------------------------------------------------------*/
                                   if(handShakingBuffer[4] == HOST_STATUS_END_PROCESSING)
    {
#if PRINT
      printf("\nHOST Requested End of Processing.\n");
#endif
                                                                                                                                  break;
    }
    else
    {
      handShakingBuffer[4] = DSP_PROCESSING_COMMAND;
    }

#if PRINT
    printf("\nNew HOST Command Read: ");
#endif

    /*-------------------------------------------------------------------------*/
    /* Process a HOST command                                                  */
    /*-------------------------------------------------------------------------*/
                                                                                                        switch(handShakingBuffer[3])
                                                                                                                                  {
      /*-------------------------------------------------------------*/
      /* Run the Internal SRAM test and return information if(error)       */
      /*-------------------------------------------------------------*/
                                                                                                                                  case ISRAM

#if PRINT
        printf("ISRAM\n");
#endif
```

```
        /*-------------------------------------------------------------------*/
        /* Attempt to write 0x55555555 to numerous locations and read same */
        /*-------------------------------------------------------------------*/
#if PRINT
        printf(" Write 5's.\n");
#endif
        if((error=mem_test(ALL_5,INTERNAL_MEM_START,INTERNAL_MEM_SIZE)) != 0)
        {
          handShakingBuffer[0]=error;
          handShakingBuffer[2]=INTERNAL_MEM_SIZE;
          handShakingBuffer[5]=ISRAM_ERR55;
        }
        else
          handShakingBuffer[5]=ISRAM_OK;

        /*-------------------------------------------------------------------*/
        /* Attempt to write 0xAAAAAAAA to numerous locations and read same */
        /*-------------------------------------------------------------------*/
#if PRINT
        printf(" Write A's.\n");
#endif
        if((error=mem_test(ALL_A,INTERNAL_MEM_START,INTERNAL_MEM_SIZE)) != 0)
        {
          handShakingBuffer[1]=error;
          handShakingBuffer[2]=INTERNAL_MEM_SIZE;
          handShakingBuffer[5] |=ISRAM_ERRAA;
        }
        else
          handShakingBuffer[5] |=ISRAM_OK;
                                                                                  brea

      /*-------------------------------------------------------------------*/
      /* Run the External SDRAM test and return information if(error)      */
      /*-------------------------------------------------------------------*/
      case SDRAM:
#if PRINT
        printf("SDRAM\n");
#endif
        /*-------------------------------------------------------------------*/
        /* Attempt to write 0x5A5A5A5A to numerous locations and read same */
        /*-------------------------------------------------------------------*/
#if PRINT
        printf(" Write 5A's.\n");
#endif
        if((error=mem_test_alt(ALT_5A,EXTERNAL_MEM_START,ext_mem_size)) != 0)
        {
          handShakingBuffer[0]=error;
          handShakingBuffer[2]=ext_mem_size;
          handShakingBuffer[5]=SDRAM_ERR5A;
        }
        else
          handShakingBuffer[5]=SDRAM_OK;

        /*-------------------------------------------------------------------*/
        /* Attempt to write 0xA5A5A5A5 to numerous locations and read same */
        /*-------------------------------------------------------------------*/
#if PRINT
        printf(" Write A5's.\n");
#endif
        if((error=mem_test_alt(ALT_A5,EXTERNAL_MEM_START,ext_mem_size)) != 0)
        {
          handShakingBuffer[1]=error;
          handShakingBuffer[2]=ext_mem_size;
          handShakingBuffer[5] |=SDRAM_ERRA5;
        }
        else
          handShakingBuffer[5] |=SDRAM_OK;
                                                                                  brea

      /*-------------------------------------------------------------------*/
```

```
    /* Run the External FLASH test and return information if(error)      */
    /*-------------------------------------------------------------------*/
    case FLASH:
#if PRINT
      printf("FLASH\n");
#endif
      /*-------------------------------------------------------------------*/
      /* Use if FLASH on Factory Board includes Checksum stored in POST  */
      /*-------------------------------------------------------------------*/
/* This test is turned off because POST changed ... it has no Checksum
#if PRINT
      printf(" CHECKSUM");
#endif
      if((error=flash_checksum(FLASH_START,POST_SIZE)) != 0)
      {
        handShakingBuffer[0]=error;
        handShakingBuffer[5]=FLASH_ERRCK;
      }
      else
        handShakingBuffer[5]=FLASH_OK;

      handShakingBuffer[1]=exp_chksum;
      handShakingBuffer[2]=cmp_chksum;

#if PRINT
       printf("\n  Expected=0x%08x\n  Computed=0x%08x\n",exp_chksum,cmp_chksum);
#endif
*/
      /*-------------------------------------------------------------------*/
      /* Attempt to write 0x55555555 to numerous locations and read same */
      /*-------------------------------------------------------------------*/
#if PRINT
      printf(" Write 5's.\n");
#endif
      if((error=flash_test(0x55,POST_END,FLASH_WRITE_SIZE)) != 0)
      {
        handShakingBuffer[0]=error;
/*        handShakingBuffer[5] |=FLASH_ERR55;    Use When Checksum first */
        handShakingBuffer[5]=FLASH_ERR55;
      }
      else
/*        handShakingBuffer[5] |=FLASH_OK;       Use When Checksum first */
        handShakingBuffer[5]=FLASH_OK;

      /*-------------------------------------------------------------------*/
      /* Attempt to write 0xAAAAAAAA to numerous locations and read same */
      /*-------------------------------------------------------------------*/
#if PRINT
      printf(" Write A's.\n");
#endif
      if((error=flash_test(0xAA,POST_END,FLASH_WRITE_SIZE)) != 0)
      {
        handShakingBuffer[0]=error;
        handShakingBuffer[5] |=FLASH_ERRAA;
      }
      else
        handShakingBuffer[5] |=FLASH_OK;

      /*-------------------------------------------------------------------*/
      /* Example of Status word of hSB - Used when FLASH test turned off */
      /*-------------------------------------------------------------------*/
//      handShakingBuffer[5]=TEST_DISABLED;
      break;

    /*-------------------------------------------------------------------*/
    /* Run the MCBSP test and return information if(error)              */
    /*-------------------------------------------------------------------*/
    case MCBSP:
#if PRINT
      printf("MCBSP\n");
```

```
#endif
#if PRINT
        printf(" Digital Loopback Mode.\n");
#endif
        if((error=mcbsp0_test()) != 0)
        {
          handShakingBuffer[0]=error;
          handShakingBuffer[5]=MCBSP_ERROR;
        }
        else
          handShakingBuffer[5]=MCBSP_OK;
        break;

      /*-------------------------------------------------------------------*/
      /* Run the TIMER test and return status                              */
      /*-------------------------------------------------------------------*/
      case TIMER:
#if PRINT
        printf("TIMER\n");
#endif
        ICR |= 0x4000;
        IER |= 0x4002;              // enable int 14 (timer0)
        CSR |= 1;
        flag = 0;
        timer0_init();
        while (flag == 0);        // wait int change flag to 1
        timer0_init();
        while (flag == 1);        // wait int change flag to 0
        IER = 0;
        handShakingBuffer[5]=TIMER_OK;
        break;

      /*-------------------------------------------------------------------*/
      /* Run the QDMA test and return information if(error)                */
      /*-------------------------------------------------------------------*/
      case QDMA:
#if PRINT
        printf("QDMA\n");
#endif
        ICR |= 0x100;
        IER |= 0x102;              // enable int 8 (qdma)
        CSR |= 1;
        flag = 0;
        src = 0x80000000;
        dst = 0xf000;
        size = 500;
        qdma_start(src,dst,size);
        while (flag == 0);        // wait int change flag to 1
        error = qdma_check(src,dst,size);
        if(error != 0)
        {
          handShakingBuffer[0]=error;
          handShakingBuffer[5]=QDMA_ERROR;
        }
        else
          handShakingBuffer[5]=QDMA_OK;
        break;

      /*-------------------------------------------------------------------*/
      /* Run the LEDS test and return information for debug                */
      /*-------------------------------------------------------------------*/
      case LEDS:
#if PRINT
        printf("LEDS\n");
#endif
        /*-------------------------------------------------------------------*/
        /* Simultaneous menu and debug modes allow user to enter values    */
        /*-------------------------------------------------------------------*/
#if PRINT
        printf(" Flash %d times with %d msec period.\n",
```

```
        handShakingBuffer[0],handShakingBuffer[1]);
#endif
      // configure CE1 as 32 bit async interface
      *(unsigned volatile int *)EMIF_CE1 = CE1_32;// EMIF CE1 control, 32bit async
      led_blink(handShakingBuffer[0],handShakingBuffer[1],handShakingBuffer[2]);
      handShakingBuffer[5]=LEDS_OK;
      // reset CE1 to 8 bit async interface
      *(unsigned volatile int *)EMIF_CE1 = CE1_8;// EMIF CE1 control, 8bit async
      break;

    /*--------------------------------------------------------------------*/
    /* Run the CODEC test and return status                               */
    /*--------------------------------------------------------------------*/
    case CODEC:
#if PRINT
      printf("CODEC\n");
#endif
      if(handShakingBuffer[2])    /* handShakingBuffer[2]=JP1Installed */
      {
        handShakingBuffer[5]=TEST_DISABLED;
      }
      else
      {
      mcbsp0_init();
#if PRINT
      printf(" Play 1KHz tone.\n");
#endif
      if((errortone=play_codec(handShakingBuffer[0], 1)) != 0)
      {
        handShakingBuffer[0]=errortone;
        handShakingBuffer[5]=CODEC_ERRTN;
      }
      else
        handShakingBuffer[5] =CODEC_OK;

      mcbsp0_init();
#if PRINT
      printf(" Play CD/MIC input.\n");
#endif
      if((error=play_codec(handShakingBuffer[1], 0)) != 0)
      {
        handShakingBuffer[1]=error;
        handShakingBuffer[5] |=CODEC_ERRCD;
      }
      else
        handShakingBuffer[5] |=CODEC_OK;
                                                                    }                   /* Test Disabled if JP1Installed (-j)*/
                                                                                                                        brea

    /*--------------------------------------------------------------------*/
    /* Run the PGMEM test and return information for debug                */
    /*--------------------------------------------------------------------*/
                                                                                                                    case PGME
#if PRINT
      printf("PGMEM\n");
#endif
      /*--------------------------------------------------------------------*/
      /* Simultaneous menu and debug modes allow user to enter values    */
      /*--------------------------------------------------------------------*/
#if PRINT
      printf(" Write 0x%08x to 0x%08x for 0x%08x words.\n",
        handShakingBuffer[0],(EXTERNAL_MEM_START+handShakingBuffer[1]),
        handShakingBuffer[2]);
#endif
      if((error=mem_test(handShakingBuffer[0],(EXTERNAL_MEM_START+handShakingBuffer[1]),
        handShakingBuffer[2])) != 0)
      {
        handShakingBuffer[0]=error;
        handShakingBuffer[5]=PGMEM_ERROR;
      }
```

```
        else
        {
          handShakingBuffer[1]+=EXTERNAL_MEM_START;
          handShakingBuffer[5]=PGMEM_OK;
        }
                                                                                                                    brea

      /*--------------------------------------------------------------------*/
      /* If user adds new command with out a case ... it winds up here     */
      /*--------------------------------------------------------------------*/
      default:
#if PRINT
        printf("ERROR\n");
#endif
        handShakingBuffer[5]=DEFAULT_ERROR;
                                                                                                                    brea
                                                                                                                }

    /*--------------------------------------------------------------------*/
    /* Tell the HOST that DSP processing is done for this command         */
    /*--------------------------------------------------------------------*/
                                                        handShakingBuffer[4] = DSP_STATUS_OUTPUT_READY;
  }

  /*--------------------------------------------------------------------*/
  /* Wrap it up                                                         */
  /*--------------------------------------------------------------------*/
#if PRINT
  printf("\nThat's All Folks!\n");
#endif

  return(0);
}

/*--------------------------------------------------------------------*/
/* mem_test() - used to test internal SRAM and external SDRAM         */
/*--------------------------------------------------------------------*/
int mem_test (int pattern, int start_address, int size_in_word )
{
  int i;
  int temp;
  int error = 0;
  int *mem_ptr = (int *)start_address;

  for(i=0;i<size_in_word;i++)    /* write pattern to the memory */
  {
    *mem_ptr++ = pattern;
  }

  mem_ptr = (int *)start_address;

  for(i=0;i<size_in_word;i++)  /* read data back from memory */
  {
    temp = *mem_ptr++;

#if SPECIAL==ENDPNT
    if(i==0)
      printf(" Read=0x%08x at Address=0x%08x\n",temp,(mem_ptr-1));
    else if((i==size_in_word-1))
      printf(" Read=0x%08x at Address=0x%08x\n",temp,(mem_ptr-1));
#endif

#if SPECIAL==INTERV
    if(!(i%(1024*256)))
      printf(" Read=0x%08x at Address=0x%08x\n",temp,(mem_ptr-1));
#endif

    if ( temp != pattern)
    {
      error++;
```

```
#if SPECIAL==MEMTST
      if(error<25)
        printf(" Read=0x%08x at Address=0x%08x\n",temp,(mem_ptr-1));
#endif
    }
  }

  return error;
}

/*-------------------------------------------------------------------------*/
/* mem_test_alt() - used to test internal SRAM and external SDRAM          */
/*-------------------------------------------------------------------------*/
int mem_test_alt (int pattern, int start_address, int size_in_word )
{
  int i;
  int temp_read,temp_expected;
  int error = 0;
  int *mem_ptr = (int *)start_address;

  for(i=0;i<size_in_word;i++)    /* write pattern to the memory */
  {
    if(i%2)
      *mem_ptr++ = ~pattern;     /* flip alternating bits       */
    else
      *mem_ptr++ = pattern;
  }

  mem_ptr = (int *)start_address;

  for(i=0;i<size_in_word;i++)    /* read data back from memory  */
  {
    temp_read = *mem_ptr++;

#if SPECIAL==ENDPNT
    if(i==0)
      printf(" Read=0x%08x at Address=0x%08x\n",temp_read,(mem_ptr-1));
    else if((i==size_in_word-1))
      printf(" Read=0x%08x at Address=0x%08x\n",temp_read,(mem_ptr-1));
#endif

#if SPECIAL==INTERV
    if(!(i%(1024*256)))
      printf(" Read=0x%08x at Address=0x%08x\n",temp_read,(mem_ptr-1));
#endif

    if(i%2)
    {
      temp_expected = ~pattern;  /* flip alternating bits       */
    }
    else
    {
      temp_expected = pattern;
    }

    if ( temp_read != temp_expected )
    {
      error++;
#if SPECIAL==MEMTST
      if(error<25)
        printf(" Read=0x%08x at Address=0x%08x\n",temp_read,(mem_ptr-1));
#endif
    }
  }

  return error;
}

/*-------------------------------------------------------------------------*/
/* flash_page_prog() - used to program FLASH                               */
```

```
/*---------------------------------------------------------------------------*/
int flash_page_prog( char pattern, char *start_address, int page_size)
{
  int i;
  char temp;
  char *flash_ptr = start_address;

  *(char *)FLASH_ADR1 = FLASH_KEY1;
  *(char *)FLASH_ADR2 = FLASH_KEY2;
  *(char *)FLASH_ADR1 = FLASH_KEY3;

  for (i=0;i<page_size;i++)
    *flash_ptr++ = pattern;

  temp = *--flash_ptr;

  while ( temp != pattern)
  {
    temp = *flash_ptr;
  }

  return 0;
}

/*---------------------------------------------------------------------------*/
/* flash_test() - used to test external FLASH                                */
/*---------------------------------------------------------------------------*/
int flash_test( char pattern, int start_address, int size_in_byte)
{
  int i = 0;
  int error = 0;
  char *flash_ptr = (char *)start_address;

  i=0;

  while(i < size_in_byte)   /* write to flash  */
  {
    flash_page_prog ( pattern, flash_ptr, 128);
    i = i+128;
    flash_ptr = flash_ptr + 128;
  }

  flash_ptr = (char *)start_address;

  for(i=0;i<size_in_byte;i++)
  {
    if( *flash_ptr++ != pattern)
    {
      error++;
#if SPECIAL==FLATST
      if(error<25)
        printf(" Read=0x%08x at Address=0x%08x\n",*flash_ptr,(flash_ptr-1));
#endif
    }
  }

  return error;
}

/*---------------------------------------------------------------------------*/
/* flash_checksum() - used to test checksum in POST on external FLASH      */
/*---------------------------------------------------------------------------*/
/*int flash_checksum (int start_address, int size_in_byte)
{
  int i = 0, error = 0;
  int checksum,*checksum_ptr;
  char *flash_ptr = (char *)start_address;
  int temp;

  i=0;
```

```
  checksum = 0;

  while(i < size_in_byte-4)
  {
    temp = *flash_ptr++;
    temp &= 0xff;
    checksum = checksum + temp;
    i++;
  }

  checksum_ptr = (int *)flash_ptr;

  exp_chksum=*checksum_ptr;
  cmp_chksum=checksum;

  if( exp_chksum != cmp_chksum) error = 1;
//  if( *checksum_ptr != checksum) error = 1;

  return error;
}
*/

/*---------------------------------------------------------------------------*/
/* mcbsp0_init() - used to initialize McBSP0                                 */
/*---------------------------------------------------------------------------*/
void mcbsp0_init()
{
                                                volatile unsigned int temp =0;
  /* set up McBSP0                                                          */
  *(unsigned volatile int *)McBSP0_SPCR = 0;                                                                    /* reset seria
  *(unsigned volatile int *)McBSP0_PCR = 0;       /* set pin control reg.  */
  *(unsigned volatile int *)McBSP0_RCR = 0x10040; /* set rx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_XCR = 0x10040; /* set tx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_DXR = 0;
  *(unsigned volatile int *)McBSP0_SPCR = 0x12001;/*  setup SP control reg */
}

/*---------------------------------------------------------------------------*/
/* mcbsp0_test() - used to test McBSP0                                       */
/*---------------------------------------------------------------------------*/
int mcbsp0_test()
{
                                            volatile unsigned int temp =0,temp1;
  /* set up McBSP0 */
  *(unsigned volatile int *)McBSP0_SPCR = 0;         /* reset serial port     */
  *(unsigned volatile int *)McBSP0_SRGR = 0x2014004A;/* set baud rate to 1MHz */
  *(unsigned volatile int *)McBSP0_PCR = 0xA00;      /* set pin control reg.  */
  *(unsigned volatile int *)McBSP0_RCR = 0x10040;    /* set rx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_XCR = 0x10040;    /* set tx control reg.   */
                                    /* one 16 bit data/frame */
  *(unsigned volatile int *)McBSP0_SPCR = 0xC1A001;  /* setup SP control reg. */
  *(unsigned volatile int *)McBSP0_DXR = 0xAAAA;

  temp1 = *(unsigned volatile int *)McBSP0_DRR;
  temp1 = mcbsp0_read();
  mcbsp0_write(0x5555);
  temp = mcbsp0_read();
  temp &= 0xffff;

  if (temp != 0x5555)
  {
    temp = 1;
    return temp;
  }

  mcbsp0_write(0xaaaa);
  temp = mcbsp0_read();
```

```
  temp &= 0xffff;

  if (temp != 0xaaaa)
  {
    temp = 1;
    return temp;
  }

  return 0;
}

/*---------------------------------------------------------------------*/
/* mcbsp0_write() - used for write to McBSP0                           */
/*---------------------------------------------------------------------*/
void mcbsp0_write(int out_data)
{
  int temp;

  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;

  while ( temp == 0)
  {
    temp = *(unsigned volatile int *)McBSP0_SPCR & 0x20000;
  }

  *(unsigned volatile int *)McBSP0_DXR = out_data;
}

/*---------------------------------------------------------------------*/
/* mcbsp0_read() - used for read from McBSP0                           */
/*---------------------------------------------------------------------*/
int mcbsp0_read()
{
  int temp;

  temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;

  while ( temp == 0)
  {
    temp = *(unsigned volatile int *)McBSP0_SPCR & 0x2;
  }
  temp = *(unsigned volatile int *)McBSP0_DRR;

  return temp;
}

/*---------------------------------------------------------------------*/
/* qdma_start() - used to start QDMA                                   */
/*---------------------------------------------------------------------*/
void qdma_start(int src, int dst, int size)
{
  *(unsigned volatile int *)CIER = 0x100;
  *(unsigned volatile int *)QDMA_SRC = src;
  *(unsigned volatile int *)QDMA_DST = dst;
  *(unsigned volatile int *)QDMA_IDX = 0x00000000;
  *(unsigned volatile int *)QDMA_OPT = 0x41380001;
  *(unsigned volatile int *)QDMA_S_CNT = size;
}

/*---------------------------------------------------------------------*/
/* qdma_check() - used to test QDMA operation                          */
/*---------------------------------------------------------------------*/
int qdma_check( int src, int dst, int size)
{
  int i,error = 0;
  int *src_ptr = (int *)src;
  int *dst_ptr = (int *)dst;

  for (i=0;i<size;i++)
  {
```

```
    if (*src_ptr++ != *dst_ptr++) error++;
  }

  return error;
}

/*-----------------------------------------------------------------------*/
/* qdma_isr() - interrupt service routine for qdma                        */
/*-----------------------------------------------------------------------*/
interrupt void qdma_isr()
{
                                        *(unsigned volatile int *)CIPR |= 0x100;
  flag = 1;
}

/*-----------------------------------------------------------------------*/
/* led_blink() - used to blink all the LEDS on the DSK                   */
/*-----------------------------------------------------------------------*/
void led_blink(int count, int ms_period, unsigned int leds_to_light)
{
  int i;

  for (i=0;i<count;i++)
  {
    *(unsigned volatile int *)IO_PORT = 0x7000000;/* turn off three leds    */
    delay_msec(ms_period/2);
    *(unsigned volatile int *)IO_PORT = leds_to_light;/* turn on user leds */
    delay_msec(ms_period/2);
  }
  *(unsigned volatile int *)IO_PORT = 0x7000000;  /* turn off three leds    */
}

/*-----------------------------------------------------------------------*/
/* get_ioport() - used to read state of ioport                           */
/*-----------------------------------------------------------------------*/
unsigned int get_ioport(void)
{
  return(*(unsigned volatile int *)IO_PORT);
}

/*-----------------------------------------------------------------------*/
/* get_bdrev() - used to read board revision bits                        */
/*-----------------------------------------------------------------------*/
unsigned int get_bdrev(void)
{
  return(((*(unsigned volatile int *)IO_PORT)>>29)&0x7);
}

/*-----------------------------------------------------------------------*/
/* delay_msec() - used to delay DSP by user specified time in msec       */
/*-----------------------------------------------------------------------*/
void delay_msec(short msec)
{
/* assume 150 MHz CPU timer period = 4/150 MHz */
  int timer_limit = (msec*9375)<<2;
  int time_start;

  timer0_start();
  time_start = timer0_read();
  while ((timer0_read()-time_start) < timer_limit);

}

/*-----------------------------------------------------------------------*/
/* timer0_read() - used to read TIMER0 count                             */
/*-----------------------------------------------------------------------*/
int timer0_read()
{
                                                    int i;
                                        i = *(unsigned volatile int *)TIMER0_COUNT;
```

```
                                                    return i;
}

/*-----------------------------------------------------------------------*/
/* timer0_start() - used to start TIMER0                                 */
/*-----------------------------------------------------------------------*/
void timer0_start()
{
                                  *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;    /* hold the timer    */
                                 *(unsigned volatile int *)TIMER0_CTRL |= 0x200;     /* use CPU CLK/4     */
                                    *(unsigned volatile int *)TIMER0_PRD  |= 0xffffffff;/* set for 32 bit cnt*/
                                   *(unsigned volatile int *)TIMER0_CTRL |= 0xC0;      /* start the timer   */
}

/*-----------------------------------------------------------------------*/
/* timer0_init() - used to initialize TIMER0                             */
/*-----------------------------------------------------------------------*/
void timer0_init()
{
                                   *(unsigned volatile int *)TIMER0_CTRL &= 0xff3f;/* hold the timer        */
                                  *(unsigned volatile int *)TIMER0_CTRL |= 0x200; /* use CPU CLK/4         */
                                   *(unsigned volatile int *)TIMER0_PRD   = 0x200; /* set for a short period*/
                                   *(unsigned volatile int *)TIMER0_CTRL |= 0x3C0; /* start the timer       */
                                            /* enable timer0 int     */
}

/*-----------------------------------------------------------------------*/
/* timer0_isr() - interrupt service routine for TIMER0                   */
/*-----------------------------------------------------------------------*/
interrupt void timer0_isr()
{
                                       *(unsigned volatile int *)TIMER0_CTRL |= 0;  /* stop timer0     */
  if(flag == 0)
    flag = 1;                    /* toggle the flag */
  else
    flag = 0;
}

/*-----------------------------------------------------------------------*/
/* play_codec() - used to test codec operation                           */
/*-----------------------------------------------------------------------*/
int play_codec(int number, int tone_playbk)
{
  int i,j;
  int temp;

// set up control register 3 for S/W reset
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x0386);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();

// set up control register 3 for mic input

  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x0306);
```

```
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();

  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x2330);
  temp = mcbsp0_read();
  mcbsp0_write(0x0);
  mcbsp0_read();
  mcbsp0_write(0x0);
  mcbsp0_read();
  if((temp & 0xff) != 0x06)
  {
#if PRINT
    printf ("  Error in setting up control register 3.\n");
#endif
    return(3);
  }

//set up control register 4
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x0400);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();

  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x2430);
  temp = mcbsp0_read();
  mcbsp0_write(0x0);
  mcbsp0_read();
  mcbsp0_write(0x0);
  mcbsp0_read();
  if((temp & 0xff) != 0x00)
  {
#if PRINT
    printf ("  Error in setting up control register 4.\n");
#endif
    return(4);
  }

// set up control register 5
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x0502);
  mcbsp0_read();
  mcbsp0_write(0);
  mcbsp0_read();

  mcbsp0_write(0);
  mcbsp0_read();
  mcbsp0_write(1);
  mcbsp0_read();
  mcbsp0_write(0x2530);
  temp = mcbsp0_read();
```

```
  mcbsp0_write(0x0);
  mcbsp0_read();
  mcbsp0_write(0x0);
  mcbsp0_read();
  if((temp & 0xfe) != 0x2)
  {
#if PRINT
    printf ("  Error in setting up control register 5.\n");
#endif
    return(5);
  }

  if(tone_playbk)
  {
    for (i=0; i< 8; i++)
    {
      sin_table[i] = sin_table[i] & 0xfffe;
    }

    for (i=0;i<number;i++)
    {
      for (j=0;j<8;j++)
      {
        mcbsp0_write((int)sin_table[j]);
      }
    }
  }
  else
  {
    for (i=0;i<number;i++)
    {
      temp = mcbsp0_read();
      temp = temp & 0xfffe;
      mcbsp0_write(temp);
    }
  }
  return(0);
}
```