The DSP56k MP3 Player User's Manual

Raymond Jimenez

June 8, 2011

Introduction

Congratulations! You're the new owner of a DSP56k-based MP3 player. This unit was designed by Raymond Jimenez personally, with emphasis in mind on:

- audiophile-quality playback
- expandibility
- usability

To get started, please flip to the Quick Start Guide on page 3, which guides you through the basics necessary just to listen to an MP3 file.

For more information about the system, and how to use its advanced features, please see the User Guide, on page 9.

For developers or interested people, there is additional Developers Documentation on page 15, which covers the system internals.

Notable features of this system include:

- *An emphasis on data bandwidth*: all processes that could benefit from DMA/interrupts do so. We can successfully play 320kBps MP3 files, and raw WAV files are almost possible. In addition, thanks to our decoder, we can handle FLAC, OGG, and WMA files at higher bitrate than MP3.
- *An audiophile-grade outpet stage*: The analog output stage of this system was heavily based on the $\gamma 2$, designed by AMB audio. As a result, it rivals even the best sounding professional systems, easily matching other digital transports in the 1000-dollar range.
- *Integrated volume control*: Due to our use of the Wolfson WM8741 DAC chip, we have fine-grained, completely silent, completely integrated volume control. Future releases may support ReplayGain adjustment.

Contents

List of Figures

List of Tables

Part I

Quick Start Guide

Chapter 1

Quick Start Guide

Your DSP56k MP3 player comes complete with one MP3 hard drive, one main playback unit (Fig 1.1), one keypad (Fig 1.2), one display unit (Fig 1.3), and one power supply.

In order to play MP3s from the hard drive, connect the main playback unit to the keypad, display, and hard drive, and power up the hard drive using its own power supply.

Then, plug in the main unit's power and wait several seconds. The unit will boot up, read the hard drive, and will be come ready.

You can then select the song to play, using the keypad depicted in Fig 1.2.

Happy listening!

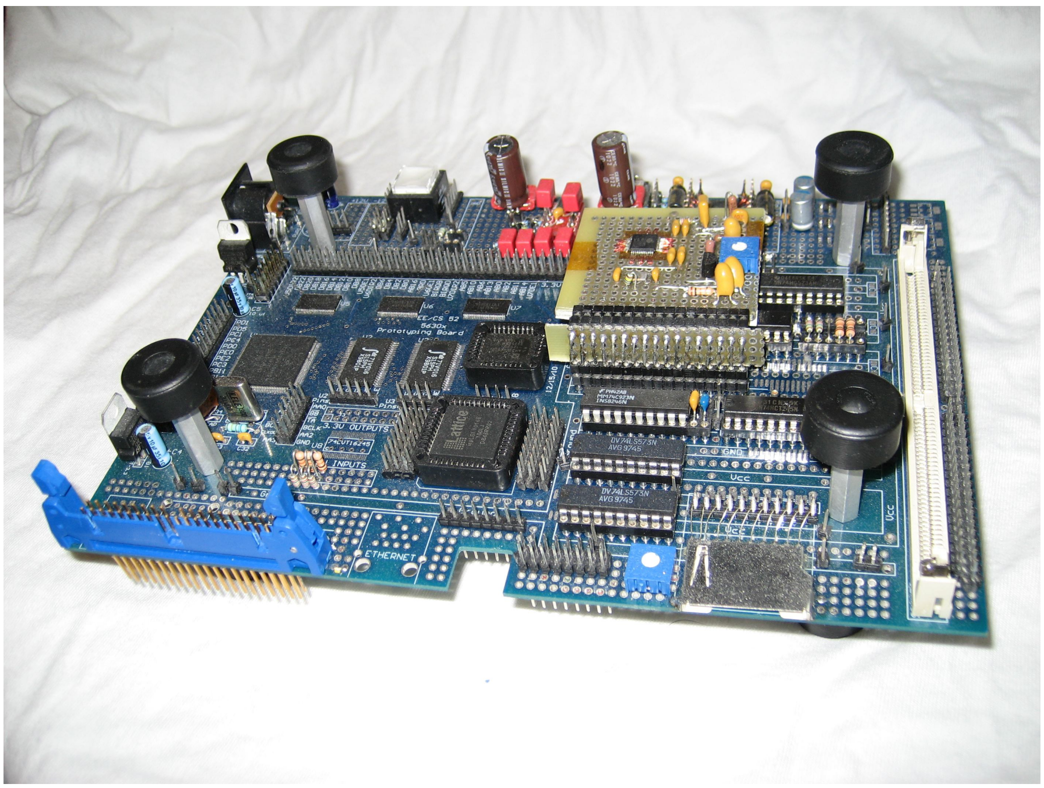


Figure 1.1: Main MP3 Playback Unit

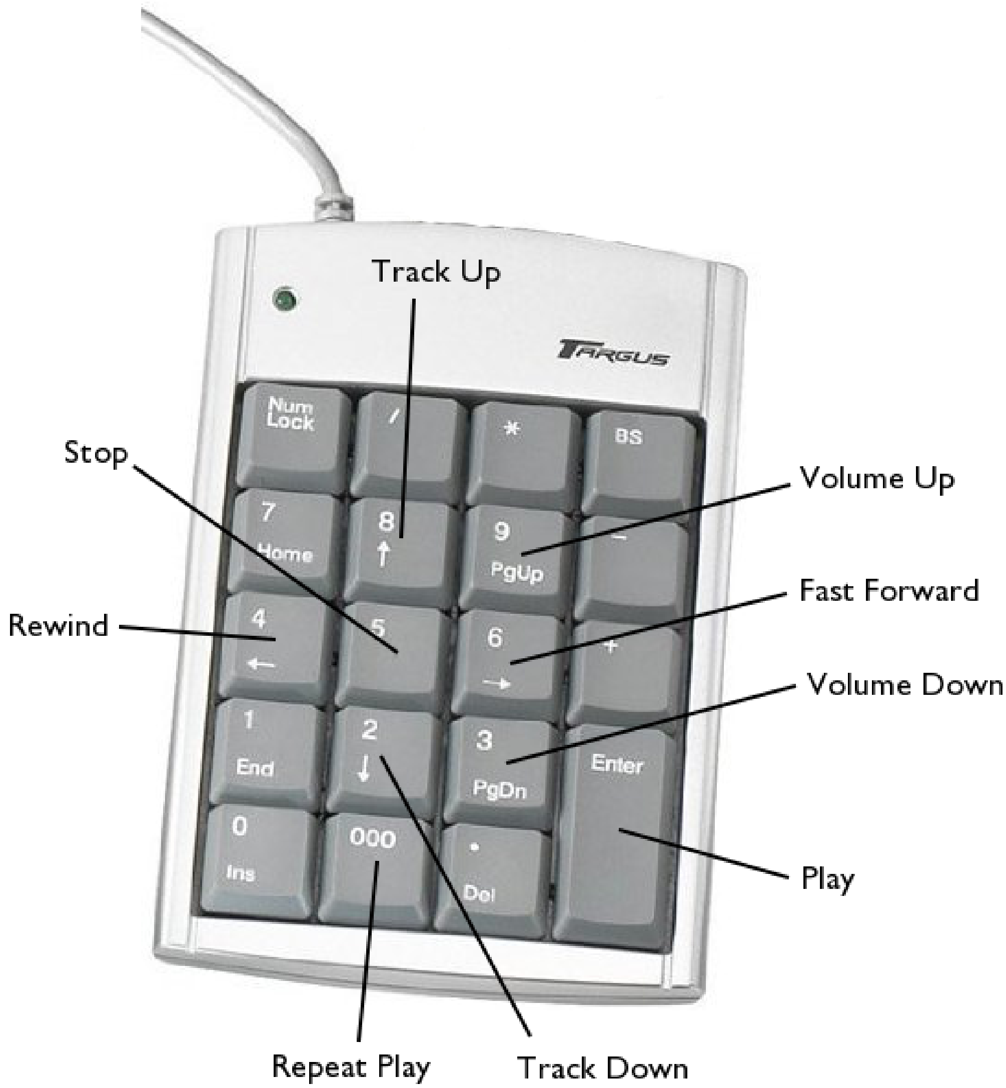


Figure 1.2: MP3 Player's Keypad

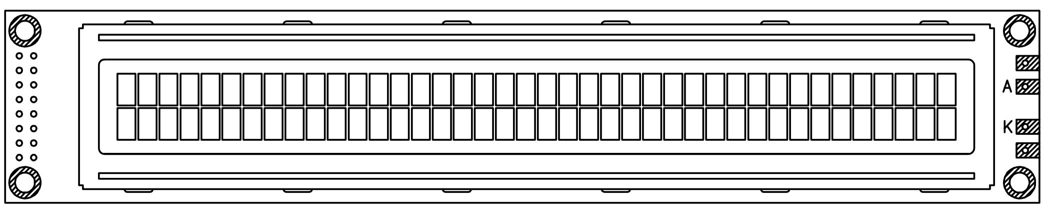


Figure 1.3: LCD Display Unit

Part II

User Guide

Chapter 2

Basic Usage

2.1 The Display

This MP3 player comes with one main display unit, a 2x40 character liquid-crystal display (LCD). At all times, the display reflects the status of the unit, in addition to its current operation and song (Fig 2.1).

1. *Song title*: The currently playing song or folder name is displayed here. Please note that if the song/folder name is too long, only the first <n> characters will display consistently; the rest may be overwritten depending on the status display.

2. *Artist*: If present, this is the current song's artist.

3. *Time*: If present, this is remaining time left in the current song.

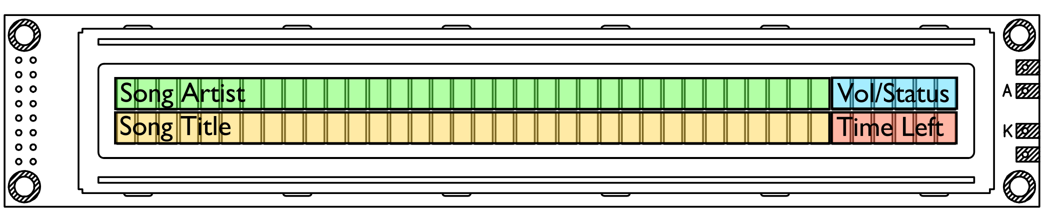


Figure 2.1: Typical Appearance of the LCD Display

4. *Status*: This indicates the unit's current status, and will be one of "Play" "Stop" "Rev" "FFwd" or "Err". An appropriate icon will also be displayed.

5. *Volume*: This indicates the current volume setting, in terms of attenuation. 0dB means the unit is at the maximum volume, while -126dB indicates that the volume is at its minimum (effectively muted). These levels are referenced to the maximum output of the unit, and not to any line-level standard.

2.2 Playing Songs

In order to play songs, simply navigate to the song desired, and press play.

If there is an entry with a preceeding ">", it is likely a folder. To navigate into a folder, press fast forward; to navigate out, press rewind.

In addition to simply playing songs, one can also put a song on repeat, by stopping the song and pressing the repeat play button.

2.3 Volume Control

In order to adjust the volume, simply press the volume up/down keys.

It may be somewhat counterintuitive, but the current volume's status is given in dB. This means that a more negative number is quieter, and a number closer to zero is louder.

Chapter 3

Advanced Features

3.1 Adding Songs

To add songs, power down the MP3 player, detach the included hard drive and attach it to your own computer, via the included USB adaptor.

Please note that we currently only support Windows computers for adding songs.

To continue, open a Run prompt and type in `cmd` .

You will see a prompt like the following:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Raymond Jimenez>
```

Then type `H:\` where H is the drive letter of the newly attached drive.

Move the music file that you would like to add to the drive, and then go back to the command prompt and type
`mksong "MP3 File.mp3" "Song Title" "Song Artist" time`
where you substitute the appropriate parameters. [1] `time` is the length of the song in seconds (e.g., a 2:40 long song is 160 seconds long).

Once this is done, remove the drive, and reconnect it to the MP3 player. You can then power up the MP3 player, and the new song should play.

---

[1] If for some reason, you receive:
```
'mksong' is not recognized as an internal or external command,
operable program or batch file.
```
please download the `mksong` utility from the EE52 website, available at .

3.2 Playing non-MP3 files

Due to the nature of our player, non-MP3 files are no different than MP3 files. In order to add a non-MP3 file, follow the steps above and the song should appear normally, available for play.

When adding the non-MP3 files, keep in mind:

- If a file's bitrate is too high, it may stutter or not play
- Only stereo 16-bit, 48000Hz audio (or less) is supported
- Currently supported formats are FLAC, WMA, AAC, and OGG.

Part III

Developer Documentation

Chapter 4

System Overview

4.1 Introduction to Architecture

A general overview of the system's architecture can be seen from Figure 4.1. Since our main purpose is to play music, not to act as a multipurpose appliance, we can make several assumptions:

1. Data should take the shortest path from the hard drive to the DAC
2. The CPU effectively shuttles data from the hard drive to the MP3 decoder
3. The CPU has only three tasks: running the user interface, shuttling data, and controlling the DAC/MP3 decoder (e.g., volume, play/stop).

The select nature of the system makes this relatively easy to implement. We first look at a detailed example of an MP3's data flow.

4.2 MP3 data flow

The MP3 data begins on the IDE hard drive. The CPU accesses the hard drive in a polling cycle, copying data from the hard drive to several buffers in DRAM. This data is then reformatted by the CPU for proper byte order. In order for the data to go smoothly from memory to the MP3 decoder, we double buffer our processed data. The MP3 decoder has the ability to tell the CPU it can handle more data (as it can decode faster than realtime, and has an onboard buffer).

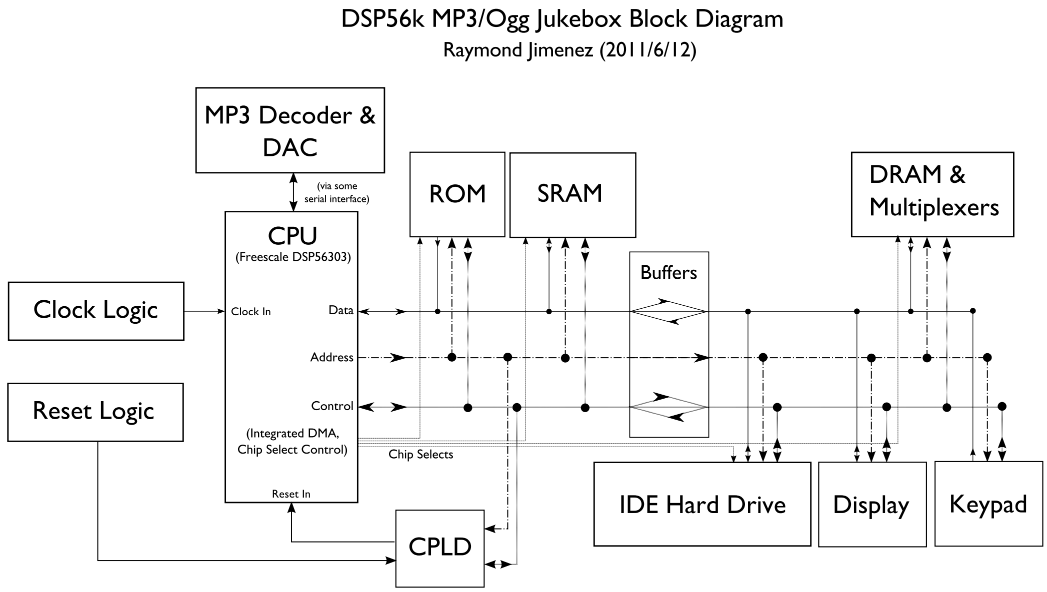


Figure 4.1: Overall System Block Diagram

When the MP3 decoder requests more data via an interrupt, we begin sending it data via a serial port, which is handled via DMA. We then begin manually fetching data in a polling loop from the hard drive, as before, to fill another buffer. We have no more than three buffers at a time (emptying, full, and filling).

Once the data is transfered to the decoder, it decompresses the file and outputs the audio data via I^2S protocol to the DAC. The DAC's outputs then go through a finishing analog stage, then out to the headphone jack.

Please note that the CPU is format-agnostic: it does not attempt to parse the audio data. This makes it possible to support playback of any decoder-supported format, including AAC, WMA, and FLAC. Additionally, the use of DMA in serial transmission allows us to playback high bitrate files, such as uncompressed WAVs.

4.3 MP3 Decoder

We currently utilize a VS1053b MP3 decoder on a separate decoder daughter-board. This allows for expandability, as well as getting around the problem

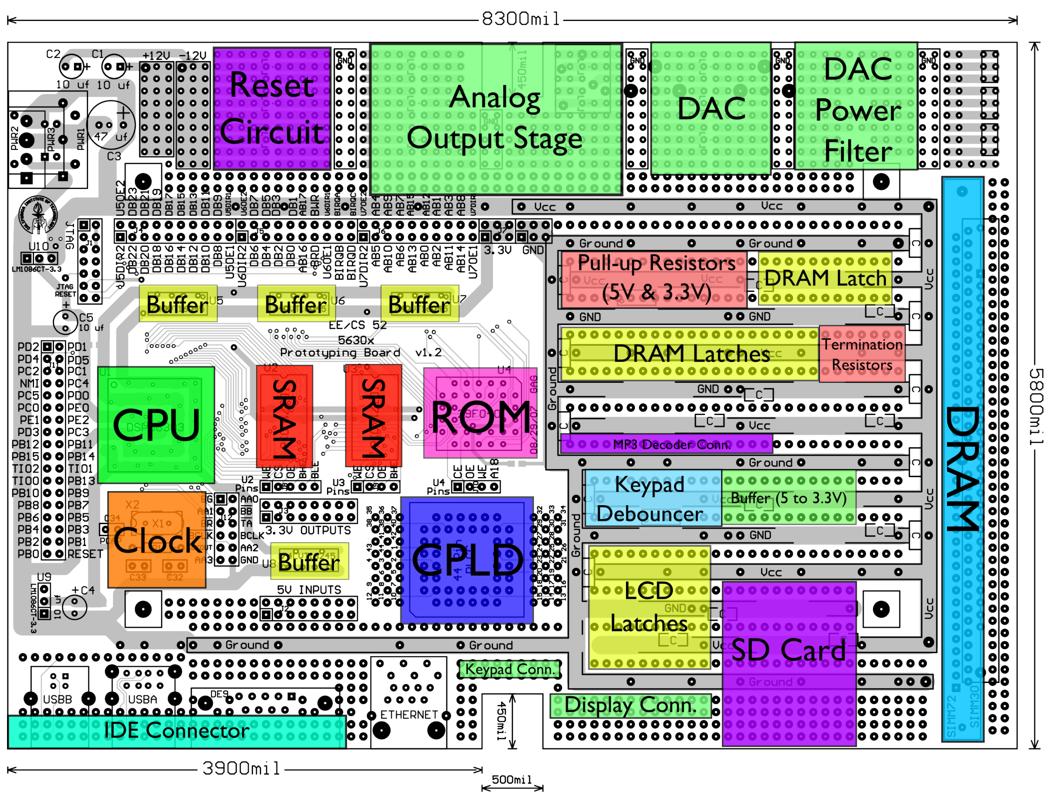


Figure 4.2: Main Board Components

Parameter	Value
Upsampling Rate	96kHz
Input Word Length	16-bit
Digital Filter	Response 3
Attenuation	-27dB
Volume Ramp	On
`MCLK`	Auto Detect
Anti-Clip Mode	On

Table 4.1: DAC Startup Mode

of board space.

4.4 DAC/Analog Stage

The DAC and analog stage have been optimized for high-quality, audiophile-grade audio output; the design is based on the $\gamma 2$ DAC from AMB Laboratories[1] with additional features for microprocessor control.

4.4.1 Power

The DAC and analog stage have separate voltage regulators in order to provide cleaner power. The +5V rail is regulated by two separate low-dropout linear regulators to +3.3V and +4.75V. The ferrite beads used in the $\gamma 2$ are absent due to space requirements.

Several μF of reservoir capacitance are provided before the regulators, and 1μF of capcitance is provided on each regulator's noise reduction pin to minimize noise.

4.4.2 DAC

The DAC is configured in 3-wire software mode, accepting commands from the CPU's second ESSI port. Upon bootup, the DAC is configured as in Table 4.1.

[1] `http://www.amb.org/audio/gamma2/`

Our MP3 decoder's maximum I^2S output is 96kHz, 16-bit stereo with a 12.288MHz master clock ($\texttt{MCLK} = 128fs$), so the input parameters were set accordingly. The digital filter was set for the best sound (response 3 is a minimum-phase, slow-rolloff filter, the same as the $\gamma 2$'s "Filter B"). The initial attenuation was set to provide a soft volume given a loud (average -5dBFS) MP3 file.

4.5 IDE

We interface with the hard drive via IDE; as the hard drive has integrated drive electronics, we don't need to worry about much. We use PIO due to the fact that we do not have an external interrupt controller, and that our main application is not multithreaded.

However, we do achieve decent transfer rates due to the fact that our code is single-threaded (interrupts take a minority of the time).

4.6 DRAM

The DRAM interface is driven mainly by the CPLD and several latches, and follows the given block diagram.

We rely on a set of three latches to hold the address given by the CPU. Whenever a read or a write occurs to DRAM, the CPLD detects this, and proceeds to:

1. assert OE on the higher row latch to output to DRAM (row address setup)
2. assert RAS to DRAM
3. deassert RAS, row latch OE
4. assert column latch OE
5. assert CAS
6. deassert CPU TA
7. wait a given interval for data ready, reassert TA

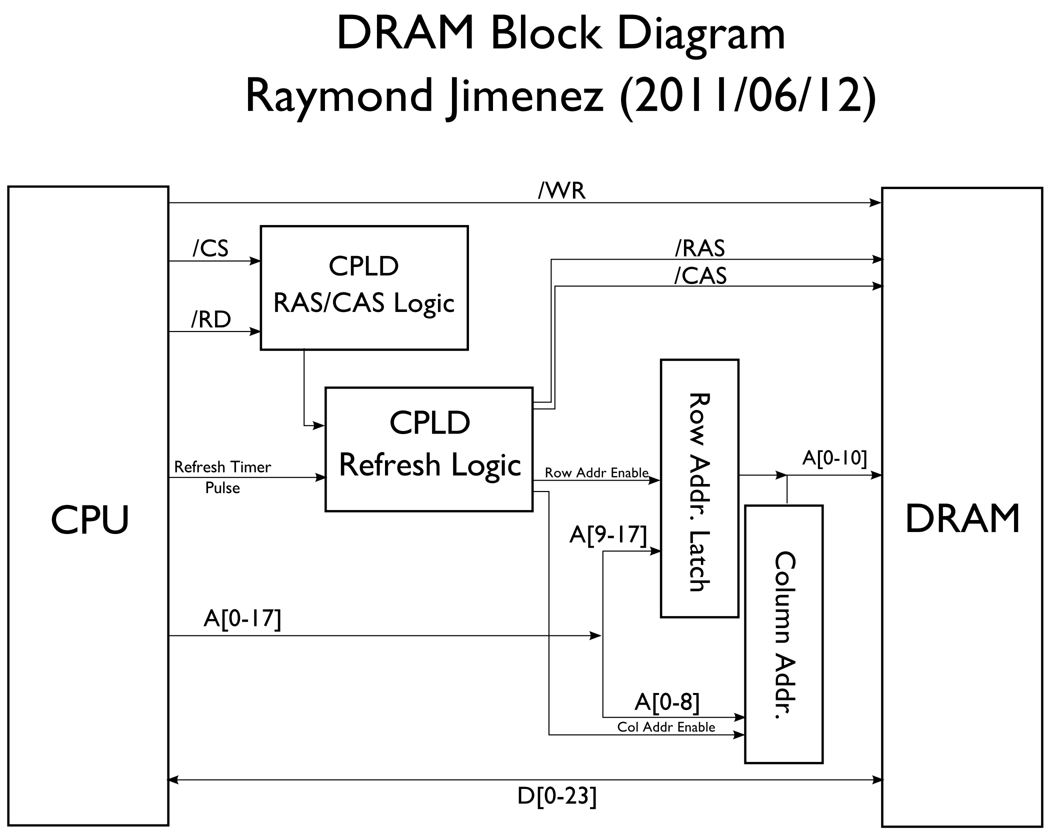


Figure 4.3: DRAM Block Diagram

8. deassert CAS, RAS, both latch OE's

Due to the fact that we use TA, the DRAM timings are specified not in the CPU but by the CPLD. For more details, please see the CPLD DRAM code on page 73.

Chapter 5

Memory Map and Registers

DSP56k memory map

Program space	Address	X space	Address	Y space	Address
	0xFFFFFF	Internal I/O	0xFFFFFF - 0xFFFF80	Internal I/O	0xFFFFFF - 0xFFFF80
Reserved	0xFF00C0				
Bootstrap ROM	0xFF0000	Internal Reserved	0xFFEFFF	Internal Reserved	0xFFEFFF
			0xFF0000		0xFF0000
	0xD3FFFF				
Boot ROM					
(256k x 8 bits)					
(forced by Mode 9)	0xD00000				
			0xA0FFFF		0xA0FFFF
		SRAM (64k)	0xA00000	SRAM (64k)	0xA00000
			0x2FFFFF		0x2FFFFF
		DRAM (1M)	0x200000	DRAM (1M)	0x200000
			0x103fff		0x103fff
		IDE controller (8k)	0x102000	IDE controller (8k)	0x102000
		Keypad (4k)	0x101FFF - 0x101000	Keypad (4k)	0x101FFF - 0x101000
		Display (4k)	0x100FFF - 0x100000	Display (4k)	0x100FFF - 0x100000
	0x004FFF		0x001BFF		
Internal 20K Program RAM	0x000000	Internal X data RAM (7k)	0x000000	Internal Y data RAM (7k)	0x000000

DSP56k memory map

Register	Value	Comments	Hex Equiv.
OMR Register:	1000 0000 0000 1000 0000 1001	- mode 9, most options turned off, no TA synchronization	0x100809
PCTL		- using 20MHz crystal, we have a multiplication factor of 4. external driven XTAL, no division, output 50% clock	0x040003
AA0:	0x100431	(general periph.)	
AA1:	0xD00609	(ROM)	
AA2:	0x200431	(DRAM)	
AA3:	0xA00831	(SRAM)	
BCR:		DSP is bus master, 1 wait state on SRAM except for area 0 (2 wait states) and 1 (7 wait states), to compensate for SRAM and DRAM	0x01d277
SR:	1100 0000 0000 0000 0000 0000	- no 16-bit mode, highest core priority, no instruction cache	0xC00000
EP:	0x000000	we set the stack to start at the bottom of internal X ram	
SZ:	0x00008F	we allocate 256 words to the stack, so = 15 + 256/2	

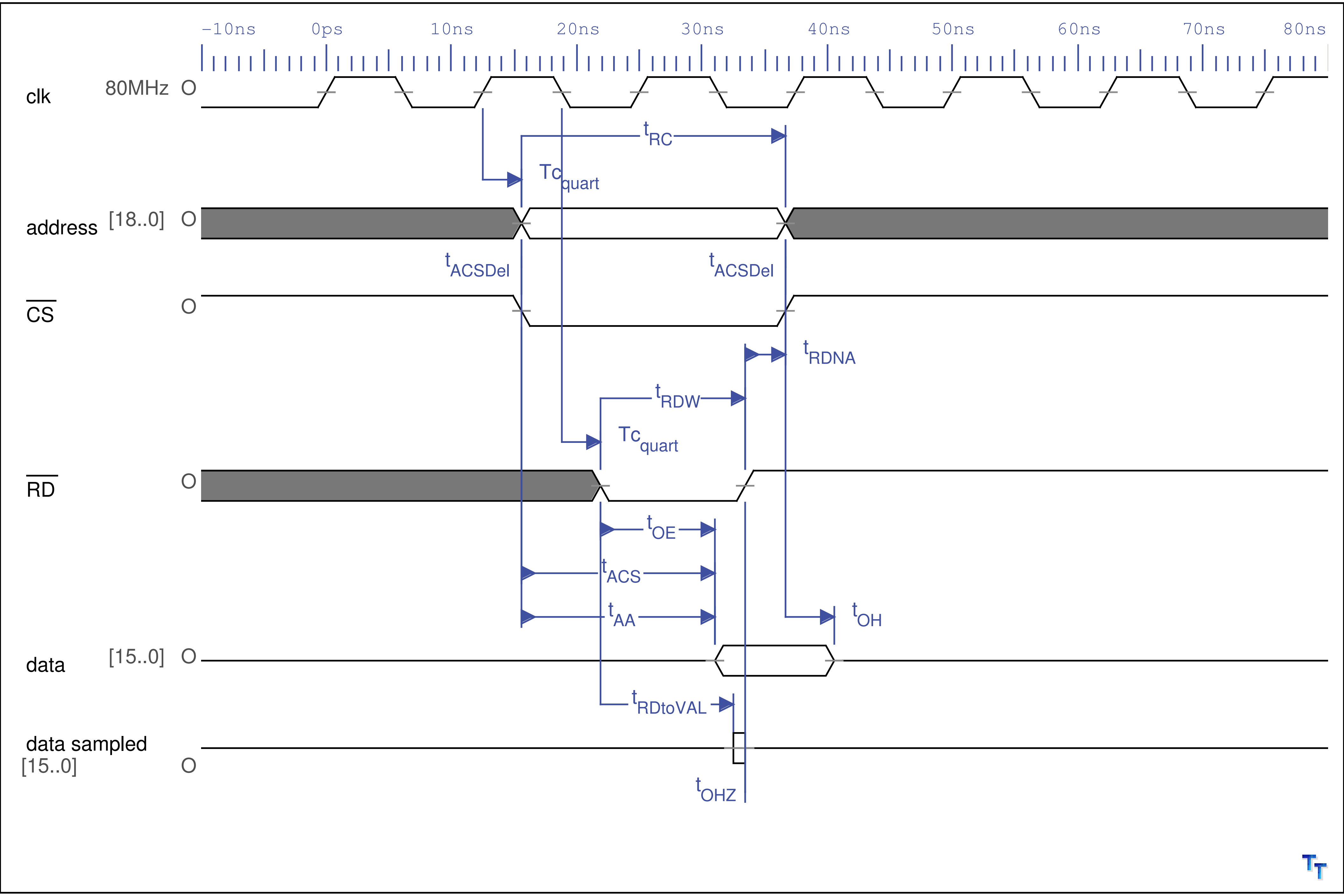
Chapter 6

Timing

There are a number of timing diagrams that we have worked on in order to make sure our circuit works. However, these timing diagrams may not reflect reality due to parasitic effects that we haven't taken into account; the wait states presented here are simply minimum settings.

6.1 Timing Targets

6.1.1 SRAM



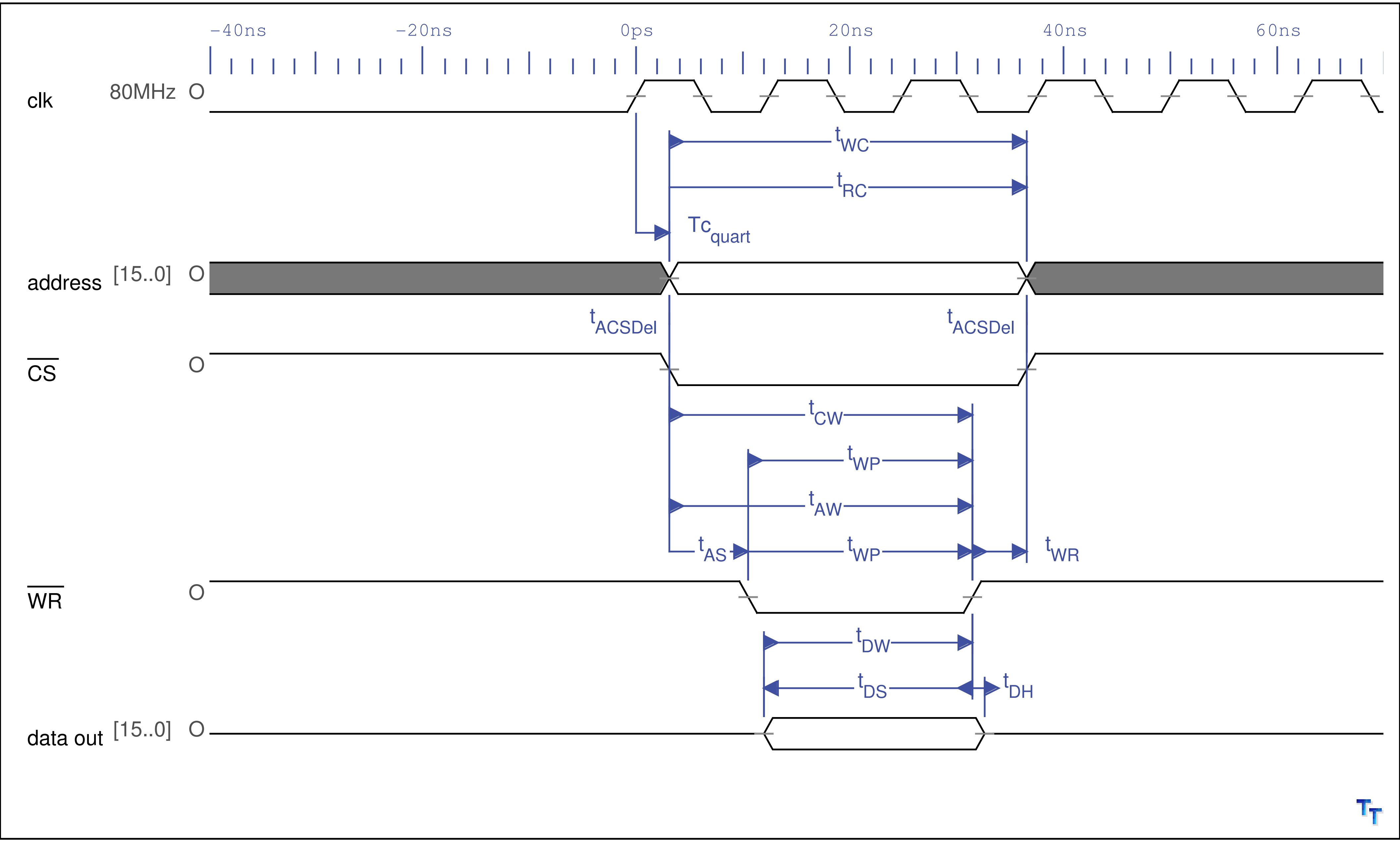
-10ns
0ps
10ns
20ns
30ns
40ns
50ns
60ns
70ns
80ns
clk
80MHz
O
tRC
Tcquart
address
[18..0]
O
tACSDel
tACSDel
CS
O
tRDNA
tRDW
Tcquart
RD
O
tOE
tACS
tAA
tOH
data
[15..0]
O
tRDtoVAL
data sampled
[15..0]
O
tOHZ

SRAM Read, CPU

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
Tc		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
Tc_{quart}	=ENG(NOM("Tc")/4)	1/4 clock period	3.125ns	3.125ns	3.125ns
Ws		wait states			1
t_{ACSDel}		address to chip select delay	0ps	0ps	1ns
t_{OHZ}		$\overline{RD}$ deassertion to data not valid (data hold time)		0ns	0ns
t_{RC}	=ENG((NOM("Ws") + 1) * NOM("Tc")-4000)	address valid width	21ns		21ns
t_{RDA}	=ENG((NOM("tRDDE")+NOM("tRDNA")+NOM("tRDW")-NOM("tRC")))	from deassertion of $\overline{RD}$ to address valid	-2.875ns	-2.875ns	-2.875ns
t_{RDDE}	=ENG((0*NOM("Tc")) + (3*NOM("Tc")/4) - 4000)	$\overline{RD}$ deassertion time	5.375ns		5.375ns
t_{RDNA}	=ENG((NOM("Tc")/4 - 2000))	$\overline{RD}$ deassertion to address not valid	1.125ns		1.125ns
t_{RDW}	=ENG(((NOM("Ws")*NOM("Tc")) + (NOM("Ws") * (NOM("Tc")/4)) - 4000))	$\overline{RD}$ pulse width	11.625ns		11.625ns
$t_{RDtoVAL}$	=ENG((NOM("Tc") * NOM("Ws")) + (NOM("Ws") * NOM("Tc") / 4) - 5000)	$\overline{RD}$ assertion to input valid	10.625ns		10.625ns

SRAM Read, SRAM

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{AA}		Address Access Time	15ns		
t_{ACS}		Chip Select Access Time	15ns		
t_{OE}		Output Enable Low to Output Valid	7ns		
t_{OH}		Output Hold from Address Change	4ns		



-40ns
-20ns
0ps
20ns
40ns
60ns
clk
80MHz
O
address
[15..0]
O
CS
O
WR
O
data out
[15..0]
O
tWC
tRC
Tcquart
tACSDel
tACSDel
tCW
tWP
tAW
tAS
tWP
tWR
tDW
tDS
tDH

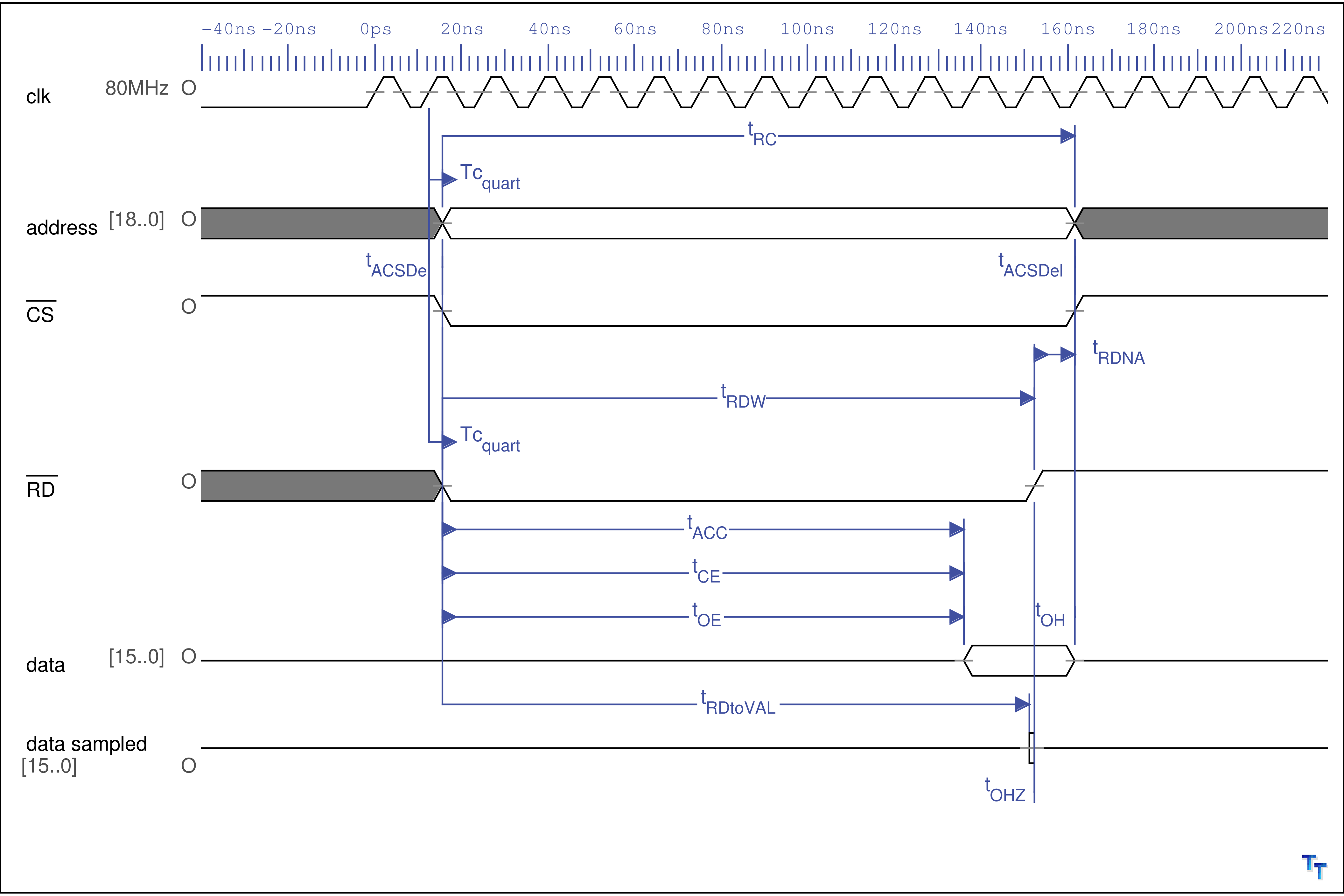
SRAM Write, CPU

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
Tc		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
Tc_{quart}	=ENG(NOM("Tc")/4)	quarter-clock period	3.125ns	3.125ns	3.125ns
t_{RC}	=ENG((NOM("Ws") + 1) * NOM("Tc")-4000)	Address valid	33.5ns	33.5ns	33.5ns
Ws		wait states			2
t_{ACSDel}		Address to chip select delay	0ps		0
t_{AS}	=ENG((3*NOM("Tc")/4) - 2000)	Address and AA valid to $\overline{WR}$ assertion	7.375ns		7.375ns
t_{WR}	=ENG((1*NOM("Tc")/4)-2000)	$\overline{WR}$ deassertion to address not valid	1.125ns		1.125ns
t_{WP}	=ENG((NOM("Ws")*NOM("Tc")) - 4000)	$\overline{WR}$ pulse width	21ns		21ns
t_{DS}	=ENG((NOM("Ws")*NOM("Tc")) - (NOM("Tc")/4) - 3000)	Data valid to $\overline{WR}$ deassertion (setup time)	18.875ns		18.875ns
t_{DH}	=ENG((NOM("Tc")/4)-2000)	Data hold time	1.125ns		1.125ns

SRAM Write, SRAM

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{WC}		write cycle time	15ns		
t_{AW}		address valid to end of write	10ns		
t_{CW}		chip select low to end of	10ns		
t_{WP}		write pulse width	10ns		
t_{DW}		data valid to end of write	7ns		

6.1.2 ROM



-40ns -20ns 0ps 20ns 40ns 60ns 80ns 100ns 120ns 140ns 160ns 180ns 200ns 220ns
clk
80MHz
O
address
[18..0]
O
CS
O
RD
O
data
[15..0]
O
data sampled
[15..0]
O
t_{RC}
Tc_{quart}
t_{ACSDel}
t_{ACSDel}
t_{RDNA}
t_{RDW}
Tc_{quart}
t_{ACC}
t_{CE}
t_{OE}
t_{OH}
$t_{RDtoVAL}$
t_{OHZ}

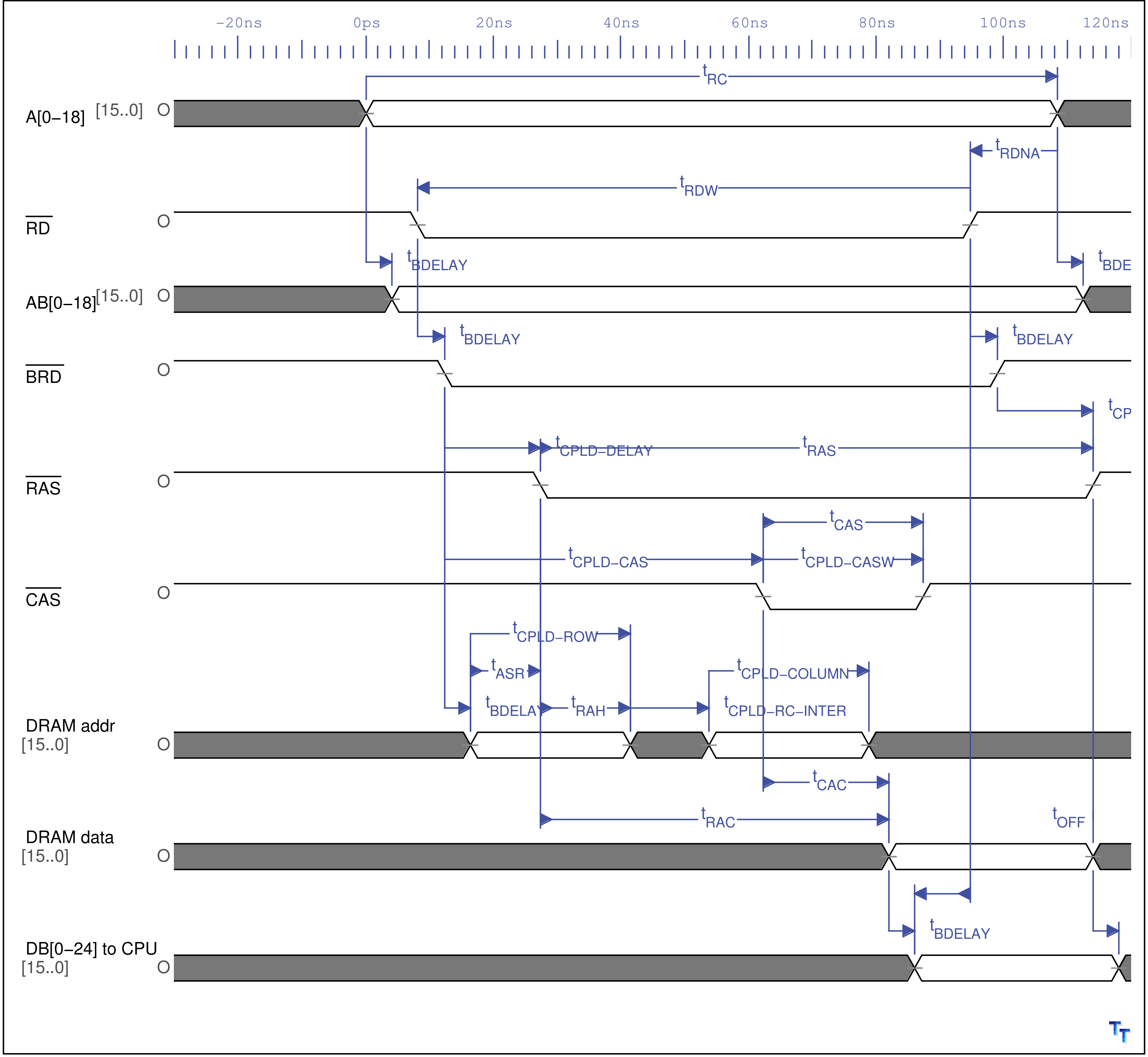
ROM Read, CPU

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
Tc		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
Tc_{quart}	=ENG(NOM("Tc")/4)	1/4 clock period	3.125ns	3.125ns	3.125n
Ws		wait states			9
t_{ACSDel}		address to chip select delay	0ps	0ps	0
t_{OHZ}		$\overline{RD}$ deassertion to data not valid (data hold time)		0ns	0ns
t_{RC}	=ENG((NOM("Ws") + 3) *	address valid width	146ns		146ns
t_{RDA}	=ENG((NOM("tRDDE")+NOM("tRDNA") +NOM("tRDW")-NOM("tRC")))	from deassertion of $\overline{RD}$ to address	14.125ns	14.125ns	14.125
t_{RDDE}	=ENG((1*NOM("Tc")) + (3*NOM("Tc")/4) - 4000)	$\overline{RD}$ deassertion	17.875ns		17.875
t_{RDNA}	=ENG(5*(NOM("Tc") /4 - 2000))	$\overline{RD}$ deassertion to address not valid	5.625ns		5.625n
t_{RDW}	=ENG(((NOM("Ws") *NOM("Tc")) + (NOM("Ws") * (NOM("Tc")/4)) - 4000))	$\overline{RD}$ pulse width	136.625ns		136.62
$t_{RDtoVAL}$	=ENG((NOM("Tc") * NOM("Ws")) + (NOM("Ws") * NOM("Tc") / 4) - 5000)	$\overline{RD}$ assertion to input valid	135.625ns		135.62

ROM Read, ROM

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{ACC}		Address to Output Delay	120ns		
t_{CE}		Chip Select Access Time	120ns		
t_{OE}		Output Enable Low to Output Valid	50ns		
t_{OH}		Output Hold from Address Change	0ns		

6.1.3 DRAM



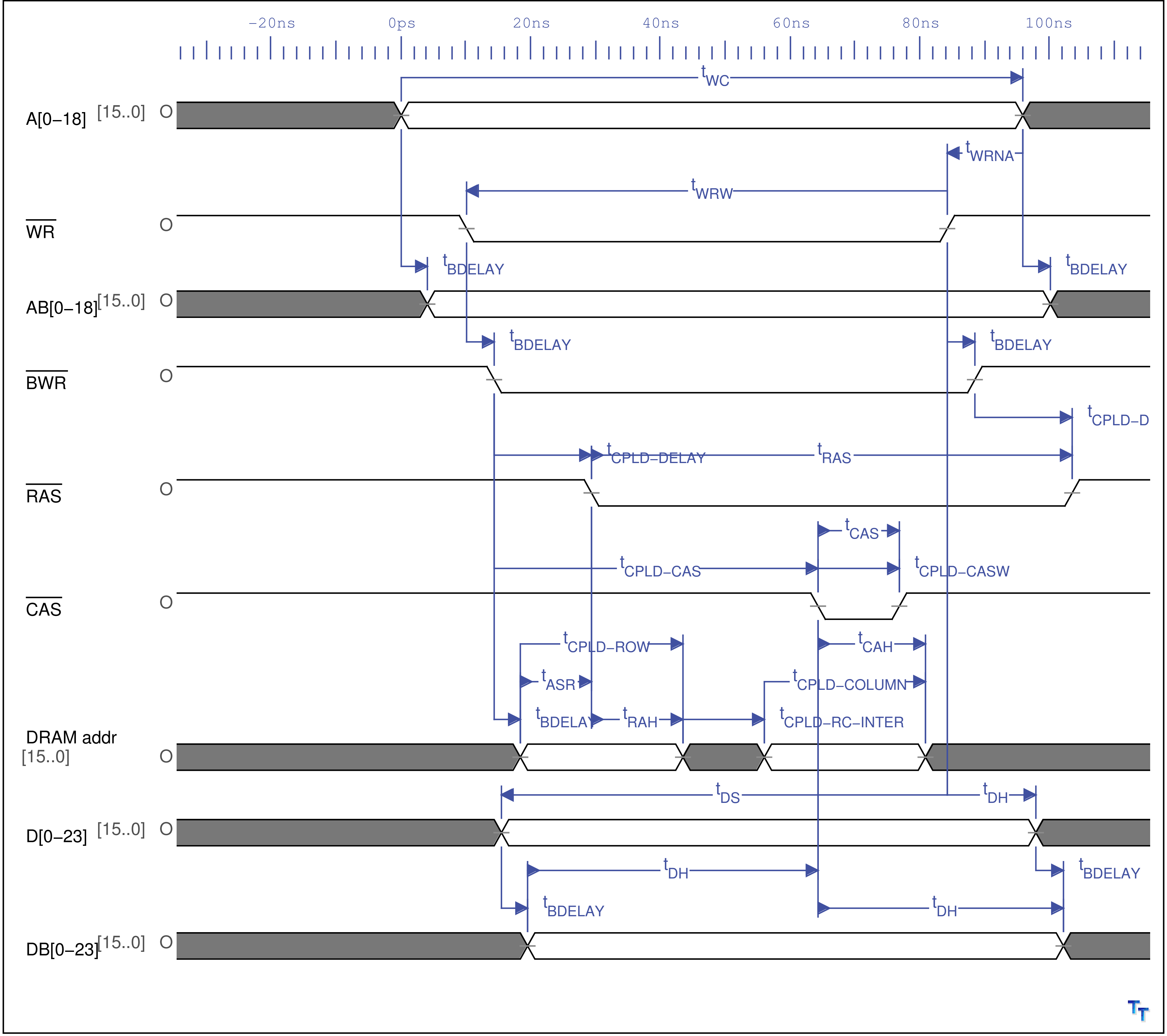
-20ns
0ps
20ns
40ns
60ns
80ns
100ns
120ns
A[0–18] [15..0] O
RD O
AB[0–18][15..0] O
BRD O
RAS O
CAS O
DRAM addr [15..0] O
DRAM data [15..0] O
DB[0–24] to CPU [15..0] O
tRC
tRDNA
tRDW
tBDELAY
tBDE
tBDELAY
tBDELAY
tCP
tCPLD–DELAY
tRAS
tCAS
tCPLD–CAS
tCPLD–CASW
tCPLD–ROW
tASR
tCPLD–COLUMN
tBDELAY
tRAH
tCPLD–RC–INTER
tCAC
tRAC
tOFF
tBDELAY

CPU timing

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{BDELAY}		general buffer delay	4.1ns		
t_{RC}		address cycle time	108.5ns		
Tc		clock period	12.5ns		
t_{RDNA}		117 - RD deassertion to address not valid	-13.625ns		
Ws		wait states	7		
$t_{CPLD\text{-}DELAY}$		average CPLD gate delay for combinatorials	15ns		
$t_{CPLD\text{-}RC\text{-}INTER}$		intervening time between row	12.5ns		
Inter cycles		#CPU cycles between row and column addr	1		
$t_{CPLD\text{-}CAS}$		time from BRD assertion to CAS assertion (CPLD synth)	50ns		
CAS cycles		#CPU cycles to lag between BRD and CAS	4		
t_{RDW}		RD assertion pulse width	-86.625ns		
$t_{CPLD\text{-}CASW}$		CAS length (CPLD synth)	25ns		
CAS length cycles		#CPU cycles for CAS length	2		
$t_{CPLD\text{-}ROW}$		row address length triggered from	25ns		
$t_{CPLD\text{-}COLUMN}$		column length triggered from	25ns		
Column cycles		#CPU cycles for column addr	2		
Row cycles		#CPU cycles for row address (CPLD synth)	2		

DRAM timing - 70ns FPM

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{RAS}		RAS pulse width	70ns	10000ns	
t_{ASR}		row address setup	0		
t_{OFF}		data hold time	0ns		
t_{RAH}		row address hold	10ns		
t_{RAC}		RAS to data		70ns	
t_{CAS}		CAS pulse width	15ns	10000ns	
t_{AA}		column addr to data		35ns	
t_{CAC}		CAS to data		20ns	



-20ns
0ps
20ns
40ns
60ns
80ns
100ns
A[0–18] [15..0] O
WR O
AB[0–18] [15..0] O
BWR O
RAS O
CAS O
DRAM addr [15..0] O
D[0–23] [15..0] O
DB[0–23] [15..0] O
t_{WC}
t_{WRNA}
t_{WRW}
t_{BDELAY}
t_{CPLD-D}
$t_{CPLD-DELAY}$
t_{RAS}
t_{CAS}
$t_{CPLD-CAS}$
$t_{CPLD-CASW}$
$t_{CPLD-ROW}$
t_{CAH}
t_{ASR}
$t_{CPLD-COLUMN}$
t_{RAH}
$t_{CPLD-RC-INTER}$
t_{DS}
t_{DH}

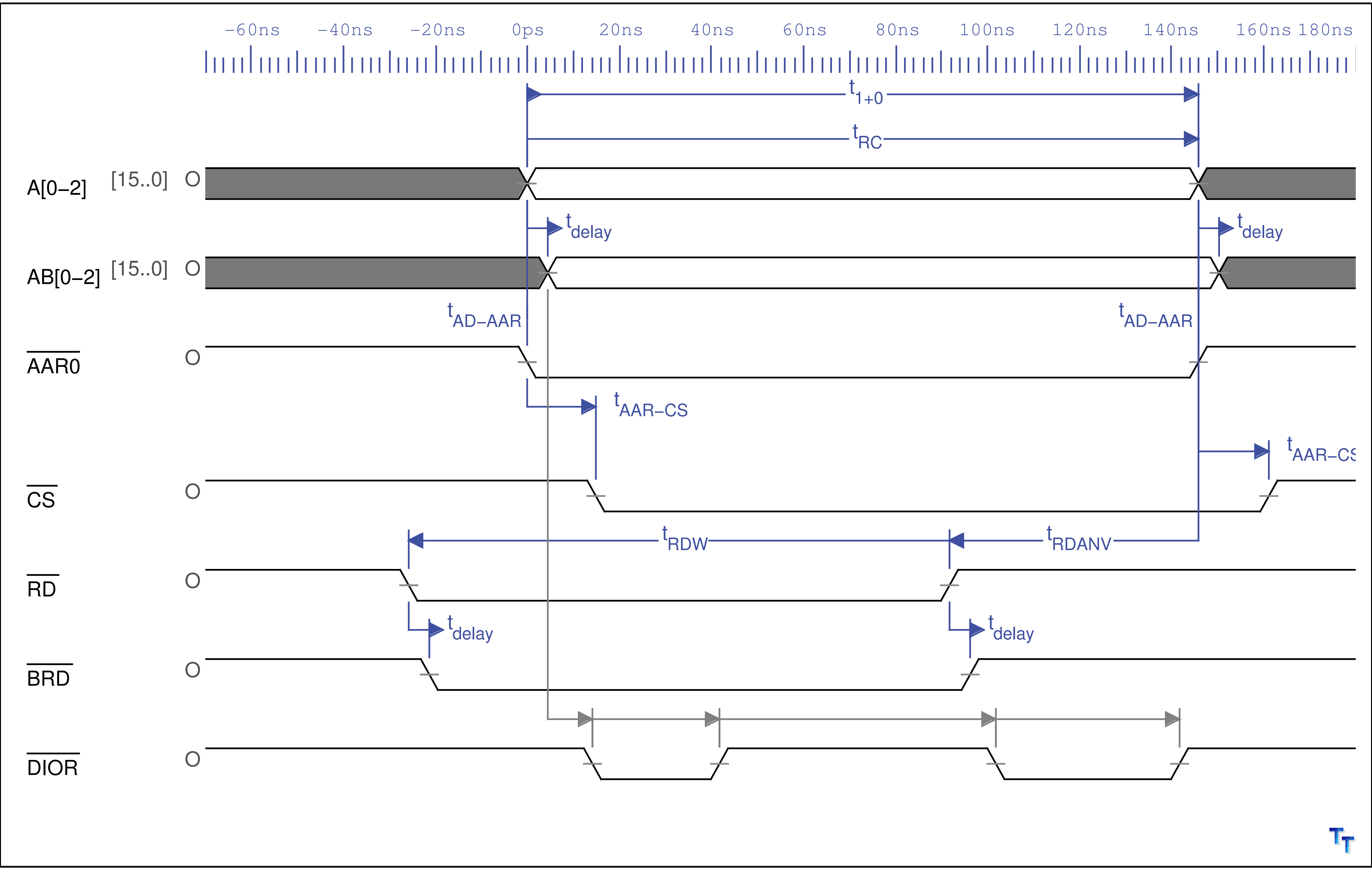
CPU timing

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{BDELAY}		general buffer delay	4.1ns		
t_{DS}		data setup time	-68.875ns		
t_{WC}		address cycle time	96ns		
t_{DH}		data hold time	13.625ns		
Tc		clock period	12.5ns		
t_{WRNA}		103 - WR deassertion to address not valid	-11.625ns		
Ws		wait states	6		
$t_{CPLD\text{-}DELAY}$		average CPLD gate delay for combinatorials	15ns		
$t_{CPLD\text{-}RC\text{-}INTER}$		intervening time between row	12.5ns		
Inter cycles		#CPU cycles between row and column addr	1		
$t_{CPLD\text{-}CAS}$		time from BRD assertion to CAS assertion (CPLD synth)	50ns		
CAS cycles		#CPU cycles to lag between BRD and CAS	4		
t_{WRW}		WR assertion pulse width	-74.125ns		
$t_{CPLD\text{-}CASW}$		CAS length (CPLD synth)	12.5ns		
CAS length cycles		#CPU cycles for CAS length	1		
$t_{CPLD\text{-}ROW}$		row address length triggered from	25ns		
$t_{CPLD\text{-}COLUMN}$		column length triggered from	25ns		
Column cycles		#CPU cycles for column addr	2		
Row cycles		#CPU cycles for row address (CPLD synth)	2		

DRAM timing - 50ns EDO

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{RAS}		RAS pulse width	50ns	10000ns	
t_{ASR}		row address setup	0		
t_{RAH}		row address hold	9ns		
t_{CAS}		CAS pulse width	8ns	10000ns	
t_{DS}		data setup	0ps		
t_{CAH}		column address hold	8ns		
t_{DH}		data hold	8ns		

6.1.4 IDE



-60ns -40ns -20ns 0ps 20ns 40ns 60ns 80ns 100ns 120ns 140ns 160ns 180ns
t_{1+0}
t_{RC}
A[0–2] [15..0] O
t_{delay}
AB[0–2] [15..0] O
t_{AD-AAR}
$\overline{AAR0}$ O
t_{AAR-CS}
$\overline{CS}$ O
t_{RDW}
t_{RDANV}
$\overline{RD}$ O
t_{delay}
$\overline{BRD}$ O
$\overline{DIOR}$ O

CPU timings

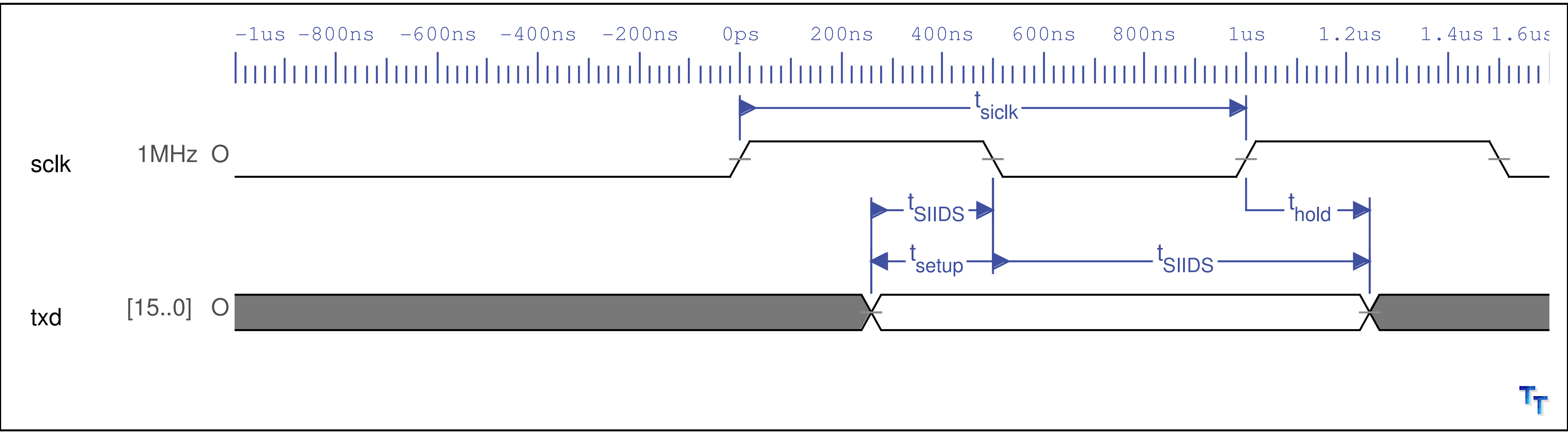
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{delay}		address buffer	4.5ns		
$t_{AD\text{-}AAR}$		delay between address bus and AAR	0	0	
$t_{AAR\text{-}CS}$		delay between AAR and CS (CPLD)	15ns		
Tc		CPU clock period		12.5ns	
Ws		Wait states	9		
t_{RDANV}		RD deassertion to address not valid	-54.25ns		
t_{RDW}		RD pulse width	-117.625ns		
t_{RC}		address pulse	146ns		

IDE timings

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_1		address valid to DIOR/DIOW setup	25ns		
t_0		cycle time	120ns		
t_{1+0}		minimum address assertion	145ns		

6.1.5 DAC (via SCI)

Please note that these timings are not actually in effect; we decided to interface via ESSI instead, which is adjustable enough that we were able to directly tune DSP parametrs to fit DAC timing.



-1us -800ns -600ns -400ns -200ns 0ps 200ns 400ns 600ns 800ns 1us 1.2us 1.4us 1.6us
sclk
1MHz O
txd
[15..0] O
tsiclk
tSIIDS
thold
tsetup
tSIIDS

CPU output

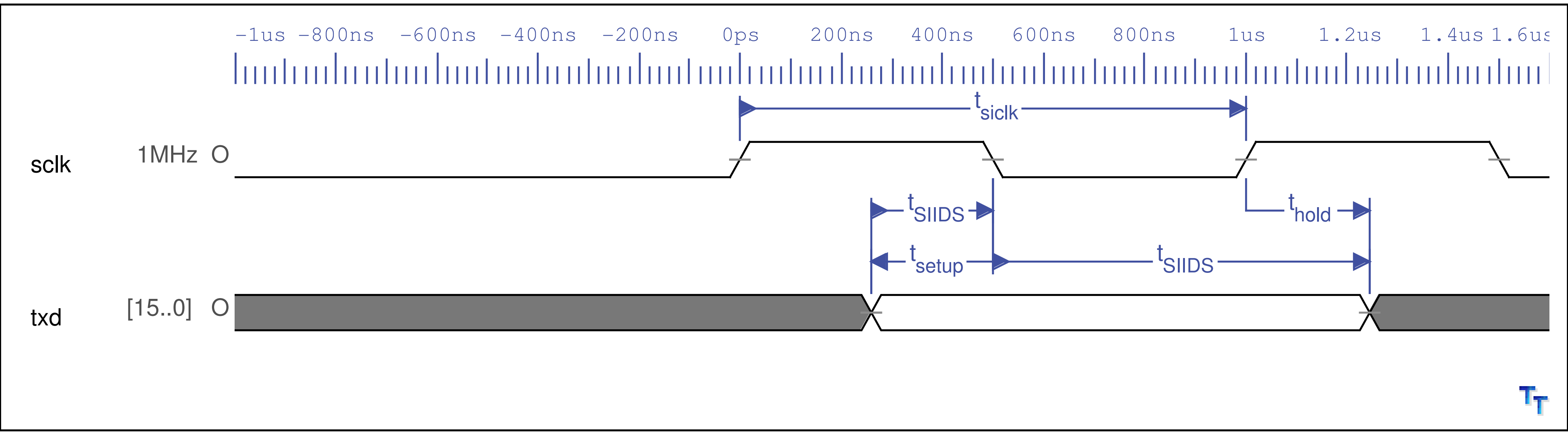
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
Tc		internal clock	12.5ns		12.5ns
t_{hold}		hold time after rising edge	243.75ns		
t_{scc}		serial clock period	1us		1us
t_{setup}		setup time before falling edge	-239.25ns		

decoder requirements

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{siclk}		clock requirement	960ns		
t_{SIIDS}		data setup time	50ns		
t_{SIIDH}		data hold time	50ns		

6.1.6 MP3 Decoder (via SCI)

Please note that these timings are not actually in effect; we decided to interface via ESSI instead, which is adjustable enough that we were able to directly tune DSP parametrs to fit MP3 decoder timing. Additionally, SCI did not support DMA, so we switched to ESSI for a speed boost.



-1us -800ns -600ns -400ns -200ns 0ps 200ns 400ns 600ns 800ns 1us 1.2us 1.4us 1.6us
tsiclk
sclk
1MHz O
tSIIDS
thold
tsetup
tSIIDS
txd
[15..0] O

CPU output

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
Tc		internal clock	12.5ns		12.5ns
t_{hold}		hold time after rising edge	243.75ns		
t_{scc}		serial clock period	1us		1us
t_{setup}		setup time before falling edge	-239.25ns		

decoder requirements

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{siclk}		clock requirement	960ns		
t_{SIIDS}		data setup time	50ns		
t_{SIIDH}		data hold time	50ns		

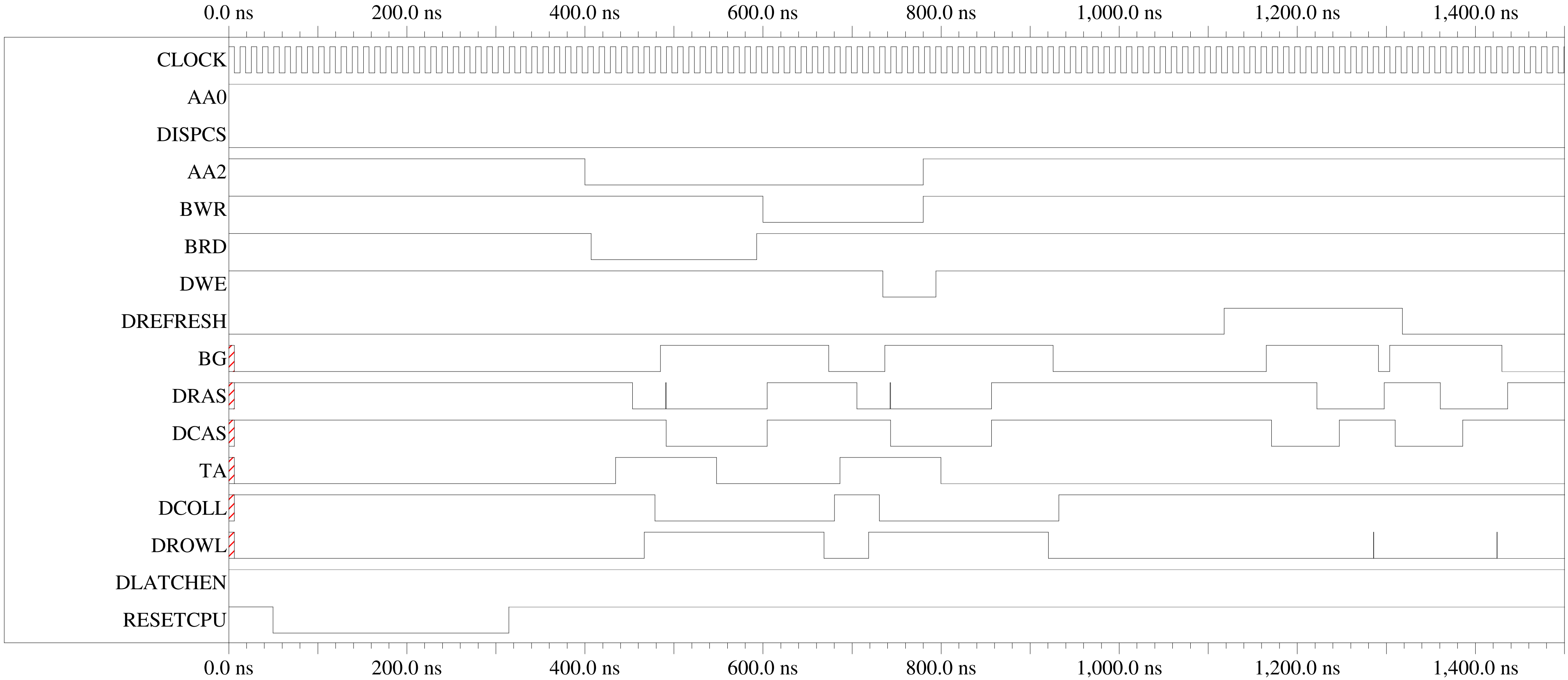
6.1.7 LCD Display

Please note that there are no current timing diagrams for the LCD display; our current design uses latches and the CPLD to interface between the LCD and the rest of the system.

6.2 Timing Simulation (CPLD)

For insurance that our CPLD design would work, we simulated several scenarios. In the following simulation, from left to right, you can see:

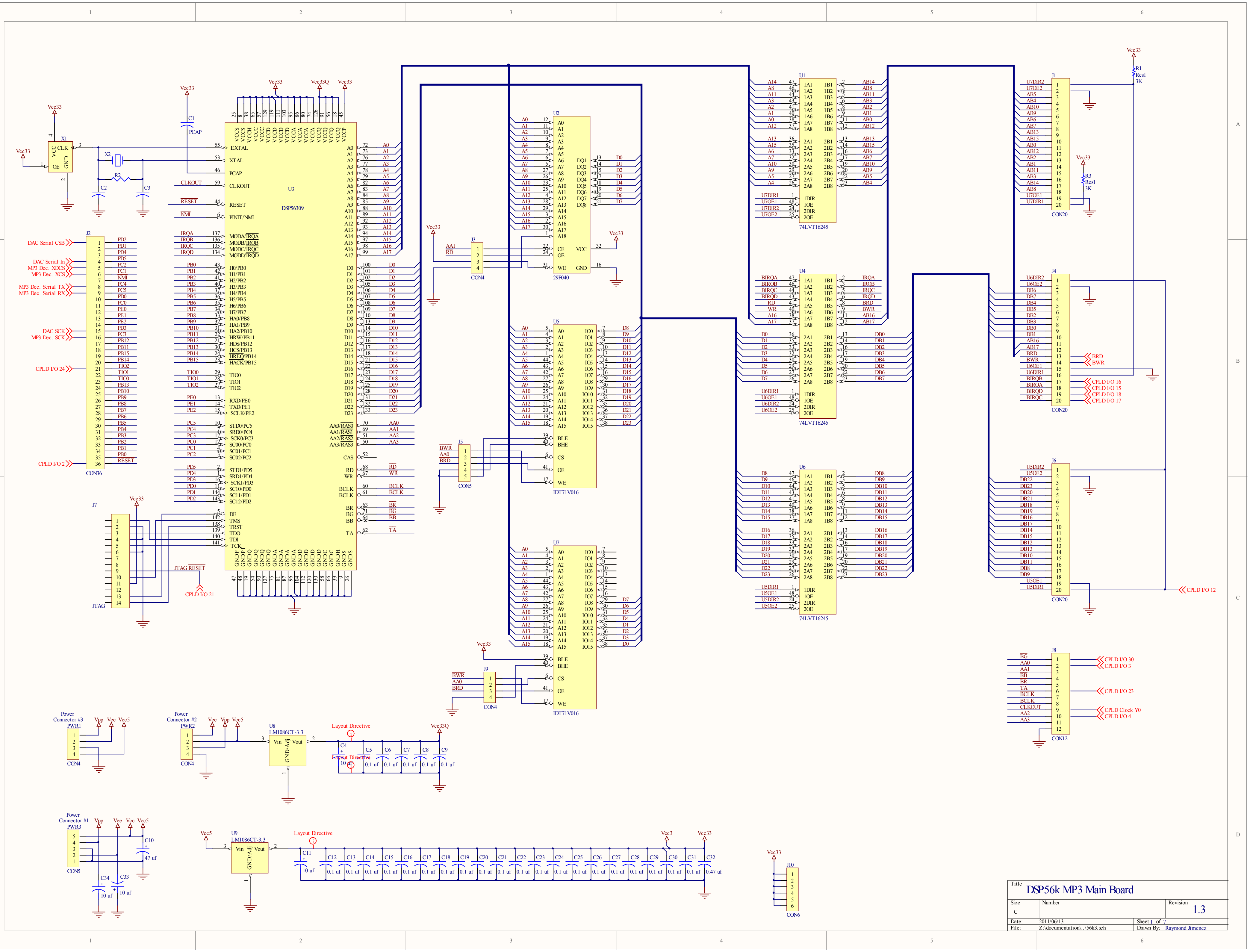
- a reset event
- a DRAM read followed immediately by a DRAM write
- two DRAM refresh cycles


0.0 ns
200.0 ns
400.0 ns
600.0 ns
800.0 ns
1,000.0 ns
1,200.0 ns
1,400.0 ns
CLOCK
AA0
DISPCS
AA2
BWR
BRD
DWE
DREFRESH
BG
DRAS
DCAS
TA
DCOLL
DROWL
DLATCHEN
RESETCPU

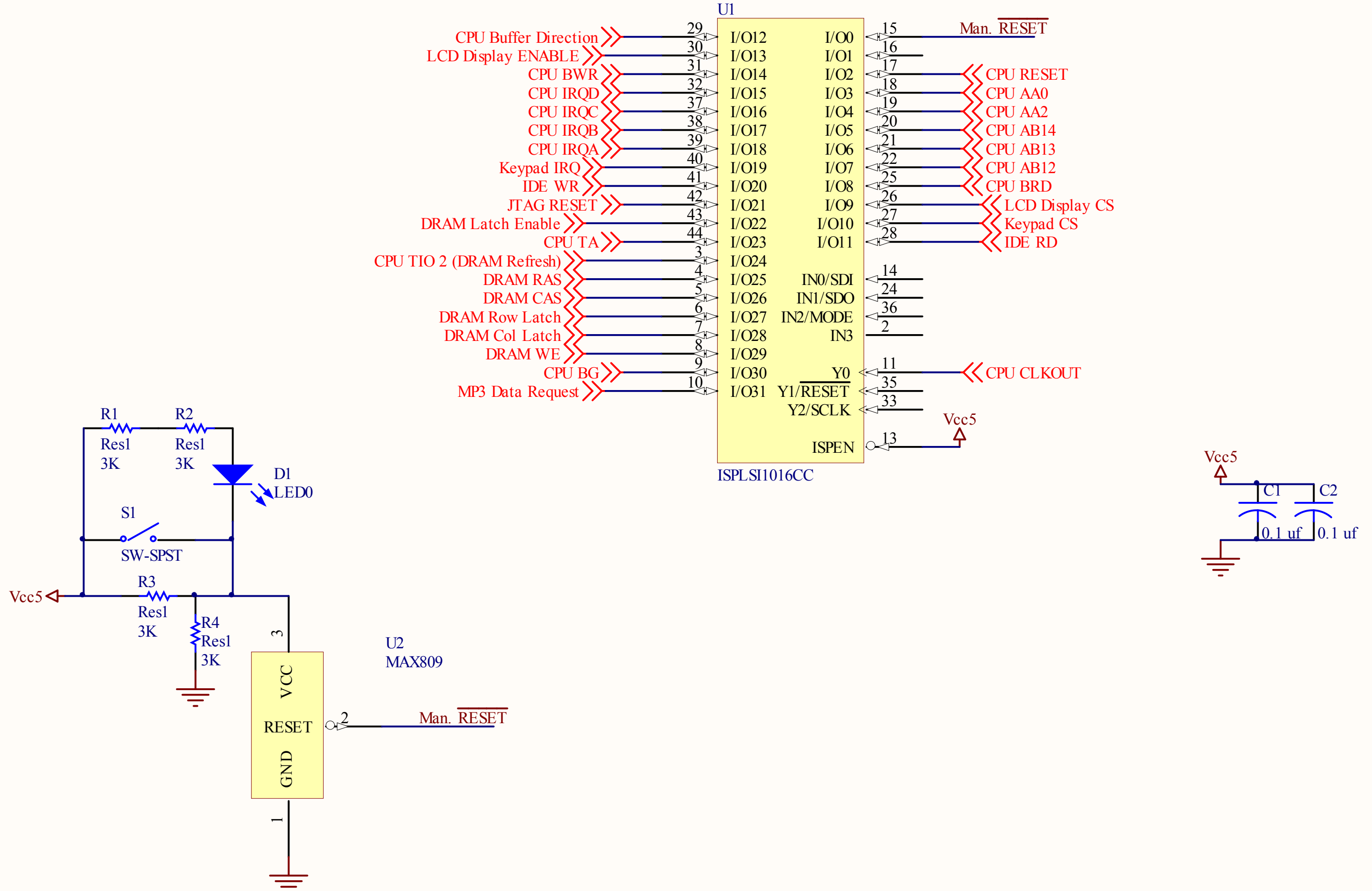
Chapter 7

Schematics

7.1 Prototyping Board/CPU

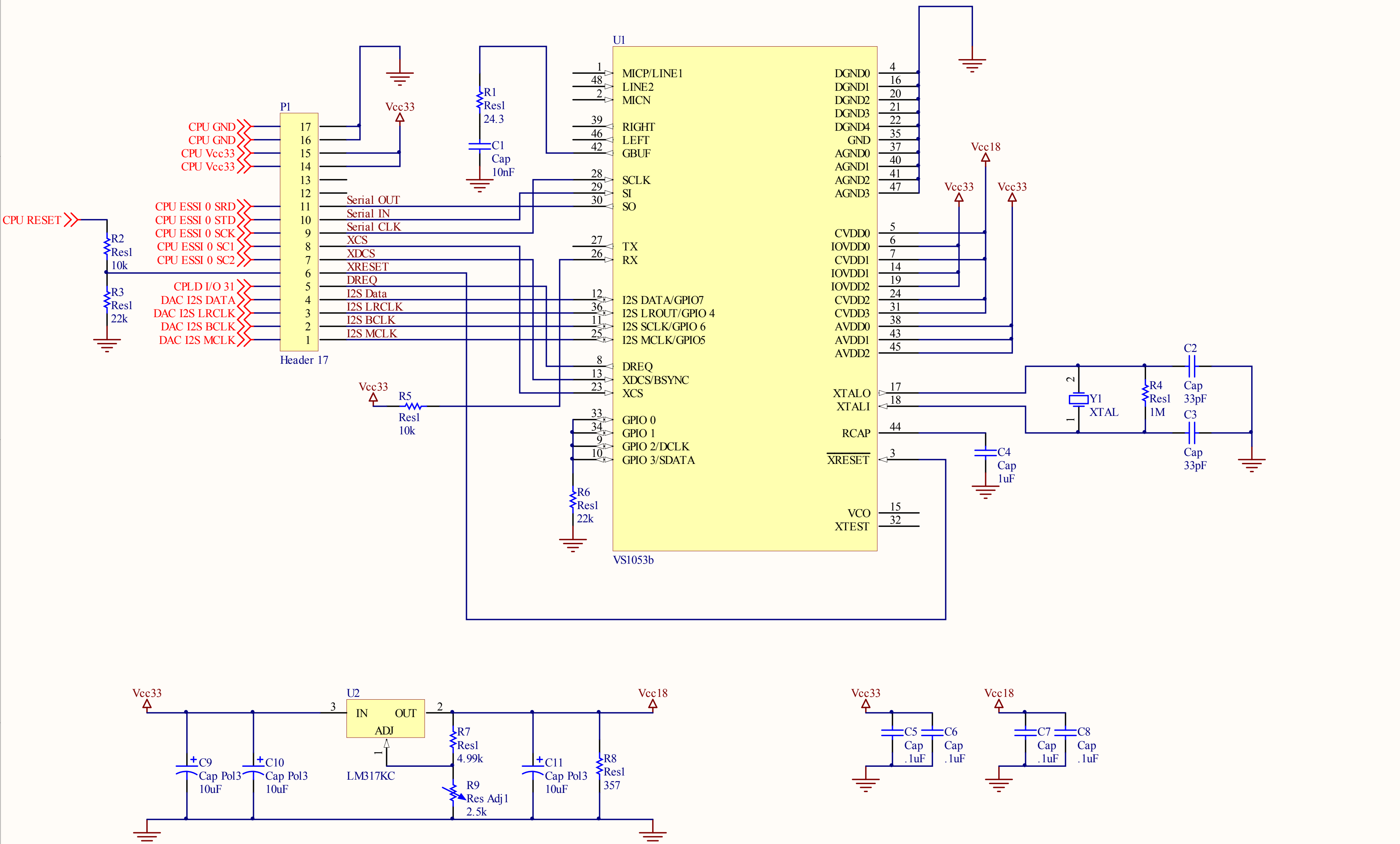

DSP56k MP3 Main Board
Size C
Revision 1.3
Date: 2011/06/13
Sheet 1 of 7
File: Z:\documentation\...\56k3.sch
Drawn By: Raymond Jimenez

7.2 CPLD



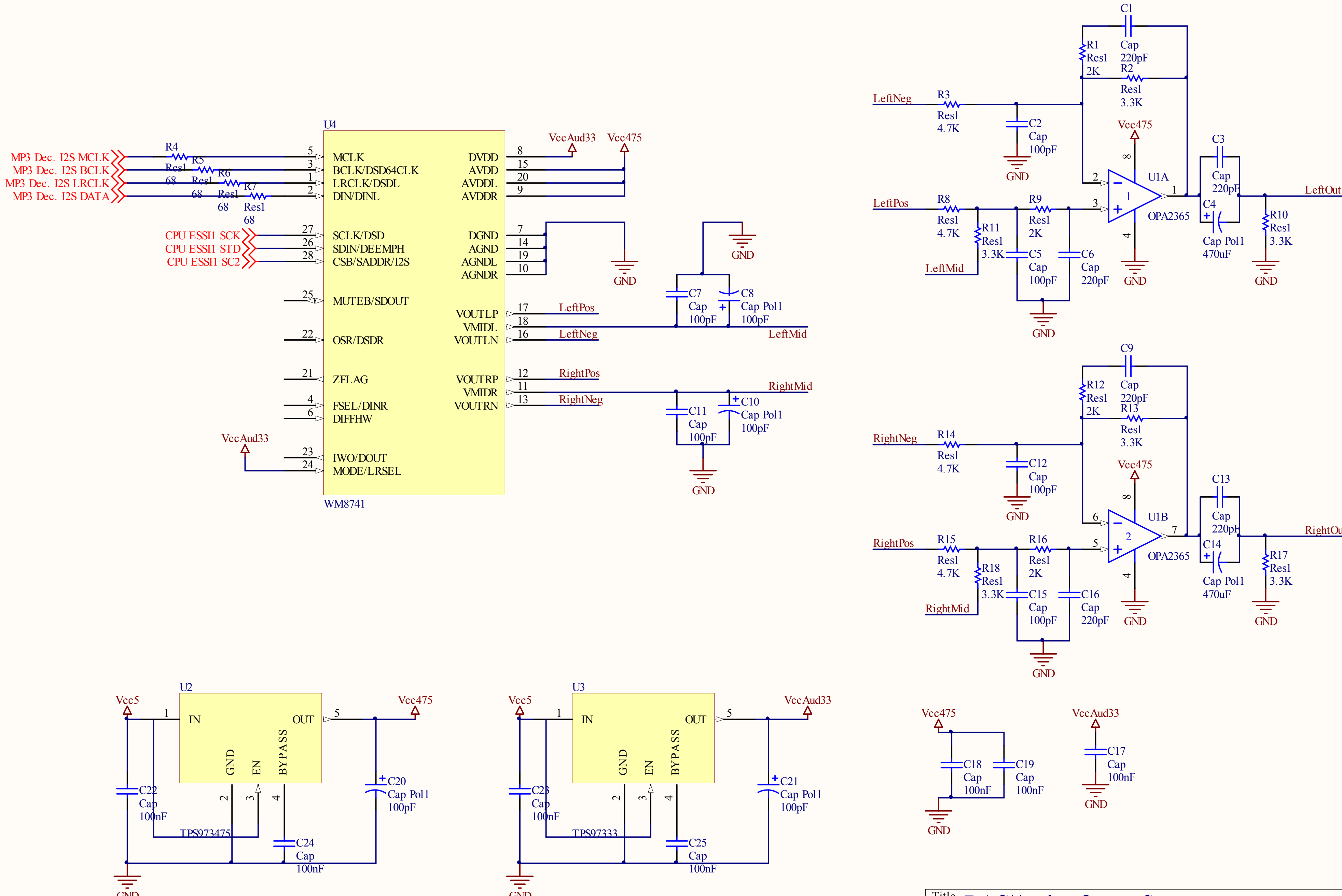
Title	CPLD/Reset Logic	
Size: Letter	Number	Revision: 1.0
Date: 2011/06/13		Sheet 2 of 7
File: Z:\documentation\..\cpld.SchDoc		Drawn By: Raymond Jimenez

7.3 MP3 Decoder Daughterboard



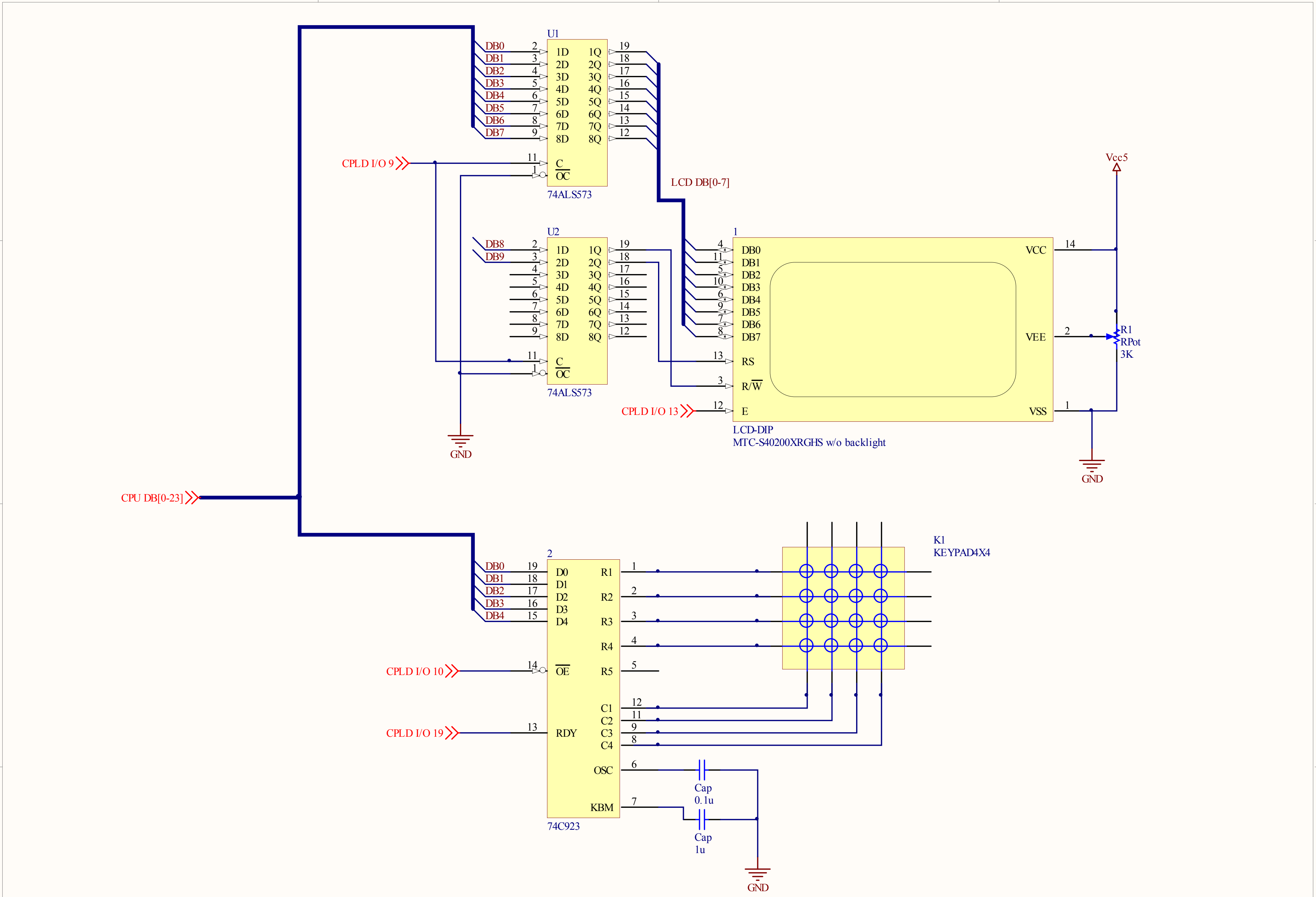
Title	MP3 Decoder Daughterboard		
Size: Letter	Number		Revision: 1.0
Date:	2011/06/13	Sheet 3 of 7	
File:	Z:\documentation\..\mp3daughter.SchDoc	Drawn By:	Raymond Jimenez

7.4 DAC/Analog Out



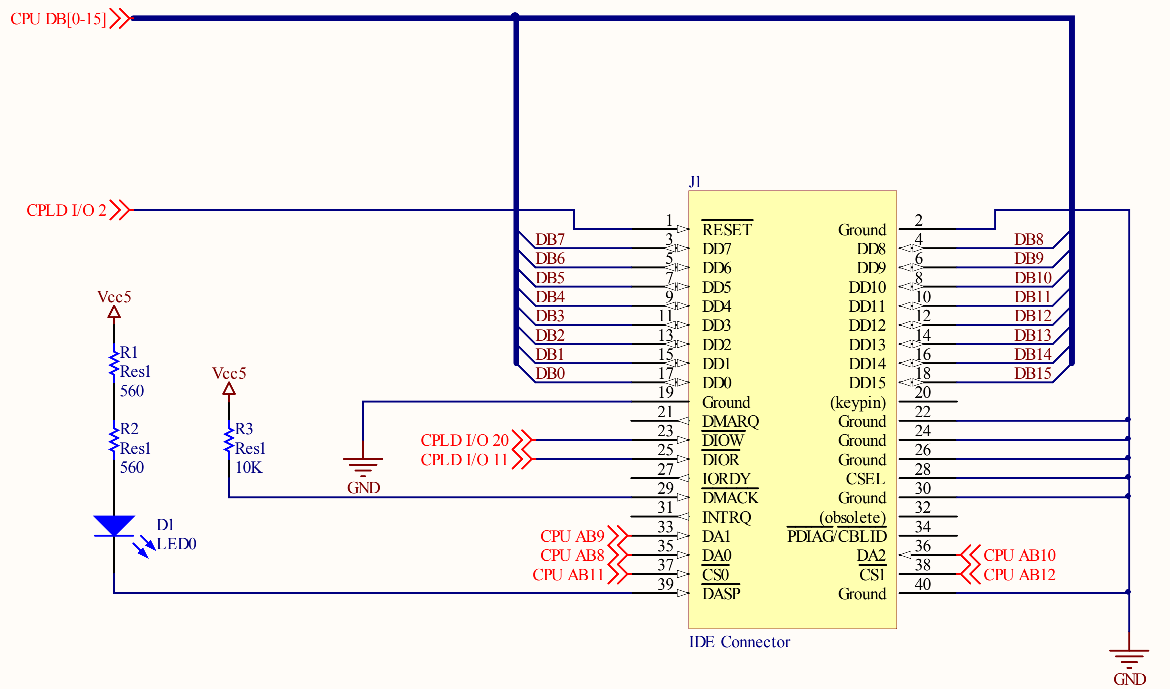
Title	DAC/Analog Output Stage		
Size: Letter	Number		Revision: 1.0
Date: 2011/06/13		Sheet 4 of 7	
File: Z:\documentation\..\analogout.SchDoc		Drawn By: Raymond Jimenez	

7.5 Display/Keypad



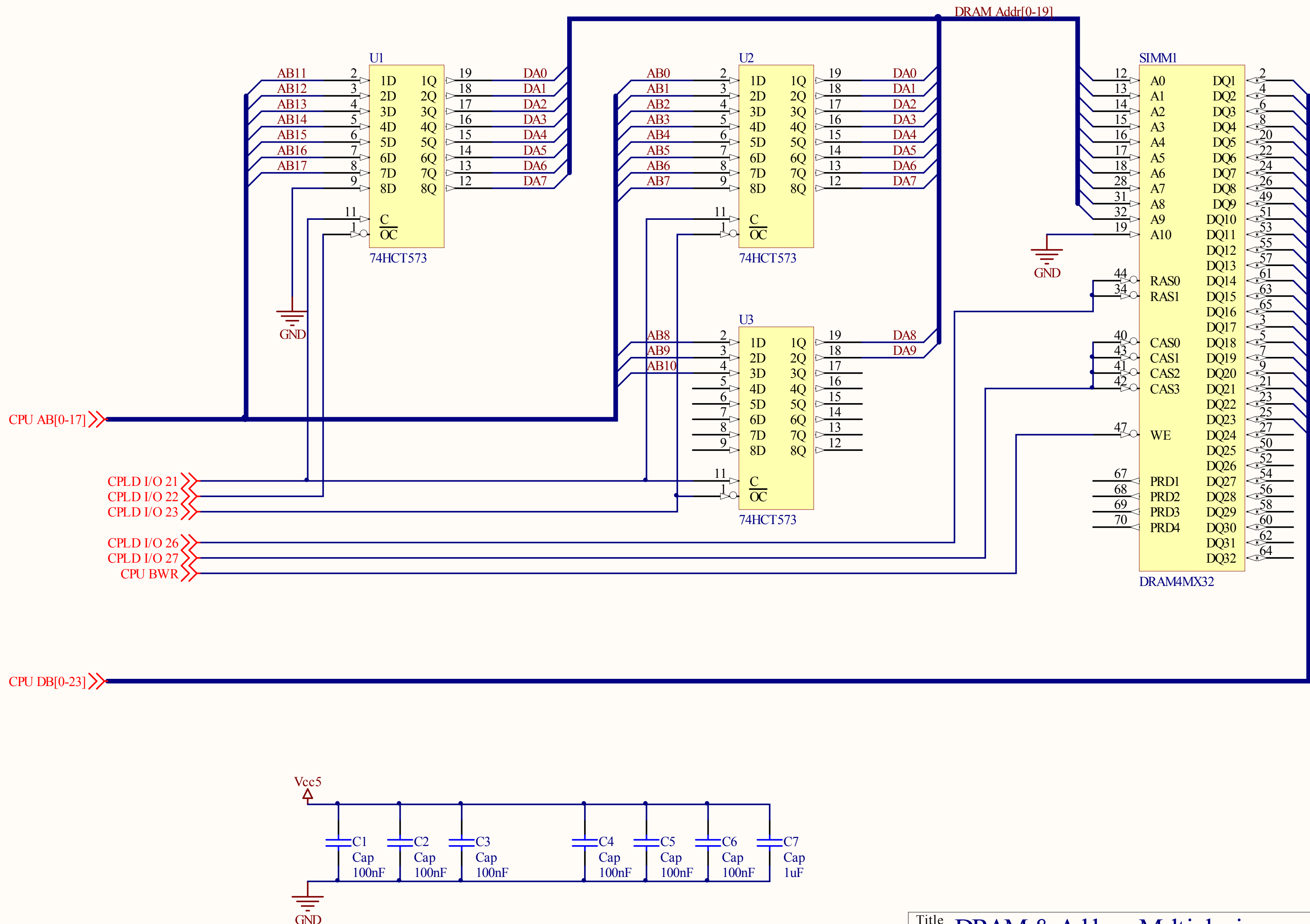
Title	LCD Display/Keypad	
Size: Letter	Number	Revision: 1.0
Date: 2011/06/13		Sheet 5 of 7
File: Z:\documentation\..\display.SchDoc		Drawn By: Raymond Jimenez

7.6 IDE Interface



Title	IDE Interface	
Size: Letter	Number	Revision: 1.0
Date: 2011/06/13		Sheet 6 of 7
File: Z:\documentation\..\ide.SchDoc		Drawn By: Raymond Jimenez

7.7 DRAM & Address Multiplexing



Title	DRAM & Address Multiplexing	
Size: Letter	Number	Revision: 1.0
Date: 2011/06/13		Sheet 1 of 1
File: Z:\documentation\..\dram.SchDoc		Drawn By: Raymond Jimenez

Chapter 8

Annotated Code

8.1 CPLD Code

The CPLD code acts as the glue that holds the system's chip select and arbitration logic together. As a result, we have several very important files:

main.abl ties everything together. It ties the logical blocks that follow with real pins; we decided to use a hierarchial approach to CPLD design because it made pin routing much easier.

reset.abl contains the reset logic, which OR's together the JTAG reset and manual reset functions, allowing both of them to reset the system.

chipsel.abl contains the chip select logic.

interrpt.abl contains logic to pass on IRQs to the CPU when neccessary, as well as controlling the DSP56309's startup mode.

dram.abl contains the DRAM refresh and RAS/CAS multiplexing logic.

```
// main glue for CPLD logic

MODULE main;

TITLE 'Main CPLD Logic';

reset interface (Reset1, Reset2 -> Reset);

chipsel interface (AA0, AA2, AB14, AB13, AB12, BRD, BWR, Clock ->
                   DispCS, KeypadCS, IDERD, IDEWR, DataDir, DispEnable);

interrupt interface (InIRQ3, InIRQ2, InIRQ1, InIRQ0, Reset, Clock ->
                     IRQ3, IRQ2, IRQ1, IRQ0);

dram interface (CS, BWR, BRD, RefreshEn, Reset, Clock ->
                TA, CAS, RAS, RowLatch, ColLatch, LatchEn, WE, BG);


DECLARATIONS

r1 FUNCTIONAL_BLOCK reset;

manreset pin 15;                  // I/O pin 0
jtagreset pin 42;                // I/O pin 21
resetcpu pin 17;                // I/O pin 2

s1 FUNCTIONAL_BLOCK chipsel;

Clock pin 11;                   // clock input Y0

AA0 pin 18;                     // I/O pin 3
AA2 pin 19;                     // I/O pin 4
AB14 pin 20;                    // I/O pin 5
AB13 pin 21;                    // I/O pin 6
AB12 pin 22;                    // I/O pin 7
BRD pin 25;                     // I/O pin 8

DispCS pin 26;                  // I/O pin 9
KeypadCS pin 27;                // I/O pin 10
IDERD pin 28;                   // I/O pin 11
IDEWR pin 41;                   // I/O pin 20
DataDir pin 29;                 // I/O pin 12

DispEnable pin 30;              // I/O pin 13

BWR pin 31;                     // I/O pin 14

i1 FUNCTIONAL_BLOCK interrupt;

IRQ3 pin 32;                    // I/O pin 15
IRQ2 pin 37;                    // I/O pin 16
IRQ1 pin 38;                    // I/O pin 17
IRQ0 pin 39;                    // I/O pin 18

KeyIRQ pin 40;                  // I/O pin 19

d1 FUNCTIONAL_BLOCK dram;
```

```
DLatchEn pin 43;                // I/O pin 22
TA  pin 44;                     // I/O pin 23
DRefresh pin 2;                 // I/O pin 24
DRAS pin 4;                     // I/O pin 25
DCAS pin 5;                     // I/O pin 26
DRowL pin 6;                    // I/O pin 27
DColL pin 7;                    // I/O pin 28
DWE  pin 8;                     // I/O pin 29
BG pin 9;                       // I/O pin 30

MP3Dreq pin 10;                 // I/O pin 31

// last pin used: I/O pin 31
// pin 16, I/O pin 1 is blown? ~\o_O/~
// pin 3, I/O pin 24 is blown? orz


EQUATIONS

r1.Reset1 = manreset;
r1.Reset2 = jtagreset;
resetcpu = r1.Reset;

s1.Clock = Clock;

s1.AA0 = AA0;
s1.AA2 = AA2;
s1.AB14 = AB14;
s1.AB13 = AB13;
s1.AB12 = AB12;
s1.BRD = BRD;
s1.BWR = BWR;

DispCS = s1.DispCS;
KeypadCS = s1.KeypadCS;
IDERD = s1.IDERD;
IDEWR = s1.IDEWR;
DataDir = s1.DataDir;

DispEnable = s1.DispEnable;

i1.Reset = r1.Reset;
i1.Clock = Clock;

IRQ3 = i1.IRQ3;
IRQ2 = i1.IRQ2;
IRQ1 = i1.IRQ1;
IRQ0 = i1.IRQ0;

// currently we have one interrupt

i1.InIRQ3 = 0;
i1.InIRQ2 = MP3Dreq;  // we use two pins for Dreq because we want to detect
                      // both edges (one to start transmits, one to stop)
i1.InIRQ1 = !MP3Dreq; // DREQ is high when the VS1053 wants data
i1.InIRQ0 = !KeyIRQ; //our IRQ pins expect to be active-low, but this one's
```

```
                    // active high per the MM74C923 datasheet.


// DRAM takes up a lot of stuff...


d1.CS = AA2;
d1.Reset = r1.Reset;
d1.BWR = BWR;
d1.BRD = BRD;
d1.RefreshEn = DRefresh;
d1.Clock = Clock;

DRowL = d1.RowLatch;
DColL = d1.ColLatch;
DRAS = d1.RAS;
DCAS = d1.CAS;
DLatchEn = d1.LatchEn;
TA = d1.TA;
DWE = d1.WE;
BG = d1.BG;


END main;
```

```
////////////////////////////////////////////////////////////////////////
//
// ABEL file for DSP56k board RESET logic
// for the Lattice ispLSI 2016E CPLD
//
// Author: Raymond Jimenez... original file from Alexander Hu/Suresh Situala
//
// Revision History:
//      April 18, 2009      Initial Revision
//      April 24, 2009      Retracted comment about pin numbers. The
//                          pin numbers you specify in this file _do_
//                          correspond to physical pin numbers in the
//                          CPLD.
//                          Changed 'datasheep' to 'datasheet'!
//              January 27, 2011    Modified to use as template
//      January 30, 2011    changed to reflect actual logic.
//
// This version inputs two active low reset signals on pin 16 and 17
// and outputs a combined active low reset signal on pin 15.
//
////////////////////////////////////////////////////////////////////////

MODULE reset;

TITLE 'Reset Logic';

// two active low reset inputs (JTAG / reset circuitry)
// You need to put in a number to replace the ?
// or else it won't work.
// Some of the pins are I/O pins, which can be put here,
// some are not, so go look at the datasheet of 1016
!Reset1     pin; //Reset input 1
!Reset2     pin; //Reset input 2


// active low reset output
// You need to put in a number to replace the ?
// or else it won't work.
!Reset      pin ISTYPE 'com' ; //Reset output





EQUATIONS

// final reset active when either of the input resets is active
Reset = Reset1 # Reset2;


END reset
```

```
/////////////////////////////////////////////////////////////////////
//
// ABEL file for DSP56k board chip select logic
// for the Lattice ispLSI 1016E CPLD
//
// Author: Raymond Jimenez... original file from Alexander Hu/Suresh Situala
//
// Revision History:
//      April 18, 2009      Initial Revision
//      April 24, 2009      Retracted comment about pin numbers. The
//                          pin numbers you specify in this file _do_
//                          correspond to physical pin numbers in the
//                          CPLD.
//                          Changed 'datasheep' to 'datasheet'!
//              January 27, 2011    Modified to use as template
//      January 30, 2011    changed to reflect actual logic.
//      February 12, 2011   added working LCD enable logic
//      February 28, 2011   added IDE DIOR/DIOW logic
//
// This file takes an AA0 line chip select input on I/O 3 (pin 18),
// and buffered address lines BA[14-12] on I/O 4-6 (pin 19-21), to
// output a chip select signal for the display (I/O 7, pin 22),
// keypad encoder chip (I/O 8, pin 23), and IDE controller (I/O 9,
// pin 24)
//
//////////////////////////////////////////////////////////////////////

MODULE chipsel;

TITLE 'Chip Select Logic';

// chip select block input from CPU
!AA0    pin; //AA0 selects 0x100000-0x1FFFFF (peripherals)
!AA2    pin; //AA2 selects 0x200000-0x2FFFFF (DRAM)

// address lines from CPU select address within that block:
AB14    pin; // buffered address 14
AB13    pin; // buffered address 13
AB12    pin; // buffered address 12

!BRD     pin; // buffered read signal
!BWR     pin; // buffered write signal


Clock    pin; // need a clock pin to work with guys

// chip select outputs
DispCS  pin ISTYPE 'com'; // display chip select; actually a active-high latch
                          // enable signal. need to work out timing to prevent
                          // glitches!
!KeypadCS pin ISTYPE 'com'; // keypad chip select
!IDEWR, !IDERD pin ISTYPE 'com'; // IDE controller pins
!DataDir pin ISTYPE 'com'; // data buffer direction pin

DispEnable pin ISTYPE 'com'; // enable pin for display, generated via delay
                // state machine.
```

```
// our address set
Addr = [AB14, AB13, AB12] ;

//state counter
S1..S0 node ISTYPE 'reg';


// display enable state counter
DispState = [S1..S0];
// we have 4 states pretty much. define using a gray code to minimize terms.
EnableSetup = [0,0];
EnablePulse1 = [0,1];
EnablePulse2 = [1,1];
EnableIdle = [1,0];

// and then we have a local counter:
DS4..DS0 node ISTYPE 'reg';
EnableCount = [DS3..DS0];


EQUATIONS

// we now need to worry about the data buffers, which switch direction
// from CPU to peripheral to vice versa.
//
// normally, the databus goes: cpu -> peripherals
// when RD asserted, it goes: cpu <- peripherals
//
// note that only a certain subset of our chips (display, keypad, IDE, DRAM)
// are after the buffers, so it's more than just AA0
//
// we define DataDir as active low to match the buffer
// wiring/direction requirement. when DataDir is asserted, the data bus goes
// from peripherals to CPU.

DataDir = BRD & (AA0 # AA2);


// we take care of general peripheral chip selects:
// only trigger display when AA0 alive and 4th digit equal 0:
// factor in BRD so that the latches see valid data guaranteed
DispCS = AA0 & (Addr == 0) & BWR;

// only trigger the keypad when AA0 alive and 4th digit equal 1:
KeypadCS = AA0 & (Addr == 1);

// we take care of IDE now. IDE is a little funny in that it has
// two CS signals and 3 address pins. our solution is to memory
// map the entire thing (CS1, CS0, DA2, DA1, DA0 in that order)
// and rely on asserting DIOW/DIOR at the right times to make sure
// it doesn't freak out. This also saves us pins.

// we rely on the IDE controllers to play nice and tristate the
// bus when we select them, but haven't pulsed DIOR/DIOW

// only trigger WR/RD to go low-high-low when the device is
```

```
// selected.
IDEWR = AA0 & ((Addr == 2) # (Addr == 3)) & BWR;
IDERD = AA0 & ((Addr == 2) # (Addr == 3)) & BRD;



// we now deal with the display enable. our display is pretty
// slow, so we do this manually.

DispState.CLK = Clock;    // we wait on the clock for wait states
EnableCount.CLK = Clock;

// definitions for timing:
// we assume an 80MHz clock (12.5ns clock cycle)
//
// our LCD requires:
// enable setup time: 40ns
// enable pulse time: 230ns
// enable deassert after pulse: 10ns
//
// translates to: 4, 19, 1 states
// we break EnablePulse into 1/2 in order to minimize the clock
// bits we need. seems to make everything run faster.
state_diagram DispState

  State EnableSetup:
    DispEnable = 0;
    IF (DispCS & BWR) THEN EnableSetup
    ELSE IF (EnableCount == 8) THEN
      EnablePulse1 WITH {EnableCount := 0;}
    ELSE EnableSetup WITH {EnableCount := EnableCount + 1;}

  State EnablePulse1:
    DispEnable = 1;
    IF (EnableCount == 15) THEN EnablePulse2 WITH {EnableCount := 0;}
    ELSE EnablePulse1 WITH {EnableCount := EnableCount + 1;}

  State EnablePulse2:
    DispEnable = 1;
    IF (EnableCount == 4) THEN EnableIdle
    ELSE EnablePulse2 WITH {EnableCount := EnableCount + 1;}

  State EnableIdle:
    DispEnable = 0;
    IF (DispCS & BWR) THEN EnableSetup WITH {EnableCount := 0;}
    ELSE EnableIdle;


END chipsel;
```

```
////////////////////////////////////////////////////////////////////////
//
// ABEL file for DSP56k board interrupt logic
// for the Lattice ispLSI 2016E CPLD
//
// Author: Raymond Jimenez... original file from Alexander Hu/Suresh Situala
//
// Revision History:
//      April 18, 2009      Initial Revision
//      April 24, 2009      Retracted comment about pin numbers. The
//                          pin numbers you specify in this file _do_
//                          correspond to physical pin numbers in the
//                          CPLD.
//                          Changed 'datasheep' to 'datasheet'!
//              January 27, 2011    Modified to use as template
//      February 12, 2011    changed from template to interrupt logic
//
//
////////////////////////////////////////////////////////////////////////

MODULE interrupt;

TITLE 'Interrupt Logic';

!IRQ0, !IRQ1, !IRQ2, !IRQ3 pin ISTYPE 'com';

!InIRQ0, !InIRQ1, !InIRQ2, !InIRQ3 pin;

S0 node ISTYPE 'reg'; // startup state counters

!Reset pin;

Clock pin; // we need to delay the IRQ via latching
           // by at least a couple of cycles.

IRQs = [IRQ3, IRQ2, IRQ1, IRQ0];

InIRQs = [InIRQ3, InIRQ2, InIRQ1, InIRQ0];

StartState = [S0];

Wait0 = [0];
Ready = [1];

Mode = [1, 0, 0, 1]; // our startup mode

EQUATIONS

StartState.CLK = Clock;
StartState.AR = Reset;

STATE_DIAGRAM StartState
  STATE Wait0: IRQs = !Mode;
    GOTO Ready;
  STATE Ready: IRQs = InIRQs;
    GOTO Ready;
```

```
END interrupt
```

```
////////////////////////////////////////////////////////////////////////
//
// ABEL file for DSP56k board DRAM controller
// for the Lattice ispLSI 2016E CPLD
//
// Author: Raymond Jimenez... original file from Alexander Hu/Suresh Situala
//
// Revision History:
//      April 18, 2009      Initial Revision
//      April 24, 2009      Retracted comment about pin numbers. The
//                          pin numbers you specify in this file _do_
//                          correspond to physical pin numbers in the
//                          CPLD.
//                          Changed 'datasheep' to 'datasheet'!
//              January 27, 2011    Modified to use as template
//      February 12, 2011    changed from template to interrupt logic
//      February 27, 2011    changed from interrupt logic to dram logic
//
//
//
////////////////////////////////////////////////////////////////////////

MODULE dram;

TITLE 'DRAM Logic';

QS2..QS0 node ISTYPE 'reg'; //setup time counter
QH2..QH0 node ISTYPE 'reg'; //hold time counter

S3..S0 node ISTYPE 'reg'; //our state machine counter

cur_writing node ISTYPE 'reg'; //we check if we are writing or not

sreg = [S3..S0];

// we define our states using a Gray code in order to minimize product
// terms.
Ready =       [0,0,0,0];
rw_rowsetup = [0,0,0,1];
rw_RAS =      [0,0,1,1];
rw_colsetup = [0,0,1,0];
rw_CAS =      [1,1,1,0];
rw_TA =       [1,1,1,1];
rw_PC =       [1,1,0,1];

CBR_CAS_LOW = [1,0,0,0];
CBR_RAS_LOW = [1,0,0,1];
CBR_RAS_HOLD = [1,0,1,1];

!CS, !Reset, !BWR, !BRD pin;
RefreshEn pin;

!CAS, !RAS  pin ISTYPE 'com';
!RowLatch, !ColLatch pin ISTYPE 'com';
LatchEn pin ISTYPE 'com';
!WE pin ISTYPE 'com';
!TA, !BG pin ISTYPE 'reg';
```

```
Clock pin;

scount = [QS2..QS0]; //counter for multiple clock cycles, setup time
hcount = [QH2..QH0]; //counter for hold timing

EQUATIONS


hcount.clk = Clock;
hcount.ar = Reset;

scount.clk = Clock;
scount.ar = Reset;

sreg.clk = Clock;
sreg.ar = Reset;

BG.clk = Clock;
BG := !S3;

LatchEn = 1;

TA.clk = Clock;

ColLatch = !RowLatch;

STATE_DIAGRAM sreg

  // we remain in the ready state until one of two things happen:
  // 1) refresh input  -     we begin to refresh
  // 2) chip select signal - we begin a read or write cycle

  // data takes priority over refresh; delaying a ref. cycle by
  // 200ns isn't going to do anything.

  STATE Ready:
    RAS = 0;
    CAS = 0;
    RowLatch = 1;
    TA := 1;
    WE = 0;

    IF (CS) then rw_rowsetup with { scount := 0; TA := 0; }
    ELSE IF (RefreshEn) then CBR_CAS_LOW with { hcount := 0; }
    //default falls through to ready


  // we purposely hold on until either BWR or BRD is selected
  STATE rw_rowsetup:
    RAS = 0;
    CAS = 0;
    RowLatch = 1;
    TA := 0;
    WE = 0;
```

```
    IF(BRD) then rw_RAS with {scount := 0}
    ELSE IF(BWR) then rw_RAS with {scount := 0}
    ELSE rw_rowsetup;


  STATE rw_RAS:
    RAS = 1;
    CAS = 0;
    RowLatch = 1;
    TA := 0;
    WE = 0;

    IF (scount == 1) then rw_colsetup with
      { scount := 0; }
    ELSE rw_RAS with {scount := scount + 1; }

  STATE rw_colsetup:
    RAS = 1;
    CAS = 0;
    RowLatch = 0;
    TA := 0;
    WE = 0;

    GOTO rw_CAS with {scount := 0}

  STATE rw_CAS:
    RAS = 1;
    CAS = 1;
    RowLatch = 0;
    WE = BWR;
    TA := 0;

    IF (scount == 4) then rw_TA with
      { scount := 0;}
    ELSE rw_CAS with { scount := scount + 1;}

  STATE rw_TA:
    RAS = 1;
    CAS = 1;
    RowLatch = 0;
    TA := 1;
    WE = 0;

    IF (scount == 3) then rw_PC with { scount := 0 }
    ELSE rw_TA with {scount := scount + 1}



  STATE rw_PC:

    RAS = 0;
    CAS = 0;
    RowLatch = 0;
    TA := 1;
    WE = 0;

    IF (scount == 5) then Ready
```

```
    ELSE rw_PC with {scount := scount + 1; }



  STATE CBR_CAS_LOW:
    TA := 1;
    RAS = 0;
    CAS = 1;
    WE = 0;

    RowLatch = 1;

    IF (scount == 3) then CBR_RAS_LOW with {scount :=0 }
    ELSE (CBR_CAS_LOW) with {scount := scount + 1}

  STATE CBR_RAS_LOW:
    TA := 1;
    RAS = 1;
    CAS = 1;
    WE = 0;

    RowLatch = 1;

    IF (scount == 1) then CBR_RAS_HOLD with {scount := 0}
    ELSE CBR_RAS_LOW with {scount := scount + 1}

  STATE CBR_RAS_HOLD:
    TA := 1;
    RAS = 1;
    CAS = 0;
    WE = 0;

    RowLatch = 1;

    IF (scount == 3) then Ready
    ELSE CBR_RAS_HOLD with {scount := scount + 1;}


END dram
```

8.2 Bootup Code (crt0.asm)

crt0.asm contains the system's bootloader; this code is executed first and copies all of the system code from RAM into ROM in order to match the compiler's expectations.

Please note that when booting via this code, program space is overwritten with ROM contents unless the ROM-copy procedures are commented out.

```
; crt0.asm
;
; This file is the initial bootup code that's executed from ROM.
; This file is heavily adopted from the standard crt0.asm
; that was included with the DSP56k developer toolkit.
;
; Perhaps most importantly, this file also incorporates the memory map
; in order to prevent the compiler from storing data in prohibited
; areas.
;
; Raymond Jimenez
; EE/CS 52
; February 16, 2011
;
; Revision history:
;
; 2011/2/16     Raymond Jimenez         initial revision
;

        opt     noidw
        opt so,xr
        page 132,66,3,3


        ;       This section should be loaded at P:0 that way reset will
        ;       cause a jump to main.

;       The reset section is located at p:0 in the mapfile. This section is
;       intended to hold system reset code. This code should jump to start
;       when it is done.

        section reset

        org     p:$0
        bra     >F__start

; see http://vnsnes.freeshell.org/dsp/c_intro.html,
; we change this to 127 in order to duplicate over all 128 event handlers
        org     p:$2
        dup     127
        bsr     >Fabort
        endm

        endsec

        section crt0

        org     x:

; ##########################################################################
; The following varaibles are used for dynamic memory allocation

; __stack_safty: Since dynamic memory and the stack grow towards each other
; This constant tells brk and sbrk what the minimum amount of space should
; be left between the top of stack during the brk or sbrk call and the end
; of any allocated memory.

```

```
        global F__stack_safety
F__stack_safety dc      1024

TOP_OF_MEMORY   equ     $a07fff
        ;this is the top of our memory in X space; anything below this is SRAM
        ; and is writable
BOTTOM_OF_MEMORY            equ     $a00000
        ;and then DSIZE is the bottom of our memory in X space.

; __mem_limit: a constant telling brk and sbrk where the end of available
;       memory is.

        global F__mem_limit
F__mem_limit    dc      TOP_OF_MEMORY


; __break: pointer to the next block of memory that can be allocated
;       The heap may be moved by changing the initial value of __break.
;       This is the base of the heap.

        global F__break
F__break        dc      TOP_OF_MEMORY

; __y_size: the base of dynamic memory.

        global F__y_size
F__y_size       dc      BOTTOM_OF_MEMORY

; errno: error type: set by some libraries

        global  Ferrno
Ferrno          dc      $0

; __max_signal the highest possible signal vector offset that might
;       be generated by the cpu.
; see http://vnsnes.freeshell.org/dsp/c_intro.html,
; we change this to $fe in order to get all the possible interrupts

        global  F__max_signal
F__max_signal   dc      $fe


; ########################################################################

        org     p:

        global  F__start
F__start

;we do our initialization here before anything else happens
; specifically: setup clock freq and chip selects

;setup PLL and clock multiplier
        movep #$040003,x:$fffffd  ; 4 * 20.000 = 80.000MHz

;map external memories per memory map
        movep #$100431,x:$fffff9  ; AA0 = general periphs
```

```
        movep #$d00609,x:$fffff8  ; AA1 = ROM (forced due to mode 9)
        movep #$200431,x:$fffff7  ; AA2 = DRAM (via our own controller)
        movep #$a00831,x:$fffff6  ; AA3 = SRAM

        jmp   end_xload

;now that we can access the ROM, unpack x-space constants down there
;judiciously adapated from the bootcode of the DSP56309 per the user manual

        move #$5400,a0          ;we fill all of x-space internals: $1bff * 3
        move #$0,a1             ;we load at the bottom of x-space
        move a1,r0              ; starting address for load
        move a1,r1              ; save it in r1
                                        ; a0 holds the number of words
        move #$d20000,r2        ;r2 is the base where we read from
        do a0,_LOOP10           ; read program words
        do #3,_LOOP11           ; Each instruction has 3 bytes
        movem p:(r2)+,a2        ; Get the 8 LSB from ext. P mem.
        asr #8,a,a              ; Shift 8 bit data into A1
_LOOP11                         ; Go get another byte.
        move a1,x:(r0)+         ; Store 24-bit result in X mem.
        nop                     ; pipeline delay
_LOOP10                         ; and go get another 24-bit word.
                                ; x-space load from EPROM done
end_xload:

;now set bus control register to set wait states
        movep #$01d27f,x:$fffffb  ; see memory map

;setup the stack extensions before we do anything
        movec #$000000,EP         ;set extension to start at bottom of intern. X
        movec #$00008f,SZ         ;and we have 256 words allocated

        movec #$c00300,SR         ;we set the SR to mask interrupts; main will
                                  ; take care of unmasking them
        movec #$100809,OMR        ;and set the OMR to enable TA sync and mode 9
                                  ; as well as stack extensions in X space

        movec #$000000,EP         ;set extension to start at bottom of intern. X
        movec #$00008f,SZ         ;and we have 256 words allocated



;       To change the base of the stack, change the value loaded into the
;       stack pointer here.

        and     #$f3,mr
        move    x:F__y_size,r6          ; initialize the stack pointer

        bsr     F__init_c_vars

        bsr     Fmain           ; run user program

        bsr     Fexit           ; when exit ( ) isn't explicitly called,
                                ; call it.
        global  F__crt0_end
F__crt0_end
```

```
        stop                    ; all done

        endsec

        section time_counter

        org     x:
        global  F__time
F__time
        dc      0

        endsec

        section io_primitives

        org     p:

        nop
        global  F__send
F__send
        rts
        nop

        nop
        global  F__receive
F__receive
        rts
        nop

        endsec

; The following section, dummy_call, is used for doing command-line
; function calls on the ADS simulator.  If you aren't going to use
; those features, it can be removed.

; The host software sets up the dummy call and patches the following
; object code to do the call and to clean things up after the call.
; If the function that the user is calling from the command line
; returns a structure or union, the value of r6 is moved into r7, and
; then r6 is incremented by the size of the structure or union being
; returned.  This negative of this size is then patched into the
; second ``move #N,n6'' instruction, so that we can clean up after the
; function call.  If no structure is returned, 0 is patched in for N
; in that move.  Following this, the current PC is moved into x:(r6)
; and then r6 is incremented.  The arguments to the function are then
; placed into registers A and B, with the remaineders placed on the
; stack starting at r6 (with r6 being incremented accordingly).
; The size of the arguments plus one (for the storage of the current
; PC) will needed to be cleaned off the stack after the function call,
; so we negate that value and patch it into the first ``move #N,n6''
; instruction.  The address of the target function is then patched
; into the jsr instruction.
;
; Following this the device is started and a ``finish'' (essentially)
; is done on dummy_call (which will then call the target function).
; If the function call is interrupted by the host software, the
; ADS/simulator is left in a state where the user can step out of the
```

```
; called function (because of the prologue code which is after the jsr
; in dummy_call).
;
; The prologue code decrements r6 by the amount used by the arguments
; to the target function plus one (to move us back to where the return
; address was stored).  Then x:(r6) is moved into ssh.  Then a (the
; return value) is tested, and r6 is decremented by the size of the
; structure returned from the target function (possibly 0 if no
; structure was returned).  Finally, we return to our caller.

        section dummy_call

        org     p:

        global  dummy_call
dummy_call
        jsr     >dummy_call
        move    #-1,n6
        move    (r6)+n6
        move    #-1,n6
        move    x:(r6),ssh
        tst     a       (r6)+n6
        rts

        endsec

        section init_table
        org p:
        xdef    F__begin_init_table
F__begin_init_table
        dc (-1)

        endsec

        section cinit
        org     p:
        xref  F__begin_init_table
        global  F__init_c_vars
F__init_c_vars
        clr   a
        clr   b
        move  #$ffffff,m0
        move  m0,m1
        move  m0,m2
        move  #F__begin_init_table,r0

        do    forever,_lab1
; p:(r0)      destination start
; p:(r0+1)    source start
; p:(r0+2)    size of source block size
; p:(r0+3)    size zero initialized block

        movem p:(r0)+,r1            ; destination start
        lua   (r1)+,b
        tst   b
        nop
        brkeq
```

```
        movem p:(r0)+,r2        ; source start
        movem p:(r0)+,r3        ; size of expicitly initialized block
        movem p:(r0)+,r4        ; size of zero initialized block

        do    r3,_lab2
        movem p:(r2)+,x0
        move  x0,x:(r1)+
        nop
_lab2

        do    r4,_lab3
        move  a1,x:(r1)+
_lab3
        nop
_lab1
        rts

        endsec

;
; we reserve memory space here to prevent things from going awry.
; i.e., we block it off to the compiler to make it seem like there
; is already something using this non-existent memory.
;
; this was done in a much nicer fashion until I accidentally deleted
; this file; you'll have to live with larger granularity now.
;
; please refer to the memory map documentation for more details.
;

; Program space reservations

; reserve everything above the internal 20K program RAM,
;  nothing else is writeable in P space
        section p_memory_reserved
        org     p:$005000       ;starts at 0x005000
        ds      $FFB000         ;0x1000000 - 0x005000
        endsec

; X space reservations

;cover the peripheral memory and unaddressed memory gaps
; (also includes memory-mapped registers)
        section x_memory_reserved0
        org     x:$a10000       ;starts above the SRAM
        ds      $5f0000         ;0x1000000 - 0xa10000
        endsec

;cover half of sram for testing purposes (fake dram)
        section sram_temp_res
        org     x:$a08000       ;starts in the middle of the SRAM
        ds      $8000           ;0xa10000-0xa08000
        endsec

;cover between sram and dram
        section x_memory_reserved1
        org     x:$300000       ;starts above the DRAM
```

```
        ds      $700000         ;0xa00000 - 0x300000
        endsec

;cover the DRAM for now because it is not functional
        section dram_temp_reservation
        org     x:$200000       ;starts at 0x200000
        ds      $100000         ;0x300000 - 0x200000 (1MByte ostensibly)
        endsec

;cover the gap between DRAM and peripherals
        section x_memory_reserved2
        org     x:$104000       ;starts at 0x104000
        ds      $fc000          ;0x200000 - 0x104000
        endsec

;we can't actually store anything in peripheral memory, so
; block it off.
        section gen_periph
        org     x:$100000       ;starts at 0x100000
        ds      $4000           ;0x104000 - 0x100000
        endsec

;cover the gap between peripherals and internal X ram
        section x_memory_reserved3
        org     x:$001c00       ;starts at 0x001c00
        ds      $fe400          ;0x100000 - 0x001c00
        endsec

;we have 256 words of stack extension at the beginning of
; X space (internal SRAM), so block it off:
        section x_mem_stack_ext
        org     x:$000000       ;at the bottom of internal X RAM
        ds      $100            ;256 words
        endsec
```

8.3 Queues (queues.asm)

queues.asm contains basic circular queue code. This code is used mainly by the display code.

```

; queues.asm
;
; This file offers several rudimentary queue (FIFO) functions.
; Note that none of these functions follow the C calling convention;
;  they are ASM-call safe only.
;
; Many of these functions are ported from EE51 HW#3.
;
; queue_init  - initializes a new queue; doesn't clear memory.
; queue_empty - checks if a queue is empty
; queue_full  - checks if a queue is full
; enqueue     - adds a word to the back of the queue
; dequeue     - grabs the word from the front of the queue
;

; Raymond Jimenez
; EE/CS 52
; February 16, 2011
;
; Revision history:
;
; 2011/2/16     Raymond Jimenez         initial revision
;


; uh, from what I see, the DSP assembler does not provide a struct construct.
; as a result, here's what we expect when we see a queue pointer:
;
; 24                 0
; +-----------------+ 0
; | head pointer    |
; |-----------------| 1
; | tail pointer    |
; |-----------------| 2
; | size (length of|
; |  data section) |
; |-----------------| 3
; |       .         |
; |       .         |
; |       .         |
; |      data       |
; |       .         |
; |       .         |
; |       .         |
; +-----------------+ 3+size
;
;
        section queues
 INCLUDE 'macros.inc'

        org     p:      ;we want to be located in program space to be in ROM


; queue_init
; Functional Specification
;
```

```
; Description:      queue_init zeros out the queue structure so that
; it can be used by other queue functions later. we assume all queues
; are in X space.
;
; Operation:        queue_init sets the head and tail pointers to be
; the same (0) and sets the size to be the passed size.
;
; Arguments:        r0 - address of queue (in X space)
;                   r1 - size of queue (total number of items - 1)
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs: None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  queue_init      ;prologue
queue_init:                     ;assembly entry point

        ;save registers we use:
        move    r3,ssh

        move    #0,r3           ;use r3 temporarily to hold 0
        move    r3,x:(r0)       ;set the head pointer to 0
        move    r3,x:(r0+1)     ;set the tail pointer to 0
        move    r1,x:(r0+2)     ;and set the size in size field

        ;restore registers we used:
        move    ssh,r3

        rts     ;return from subroutine

; queue_empty
; Functional Specification
;
```

```
; Description:      queue_empty checks whether the queue is empty, by comparing
; the head and tail pointers, and sets ZF if the queue is empty.
;
; Operation:        The function compares the head and tail pointers, and as a
; result, sets the zero flag. In other words, the cmp instruction does all of
; the work for us.
;
; Arguments:        r0 - address of the queue to look at
;
; Return Values:    ZF - set if queue is empty
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  queue_empty
queue_empty:                           ;assembly entry point

        ;we use the accumulators due to CMP, so save them:
        PUSH_ACC A
        move    x0,ssh          ;we'll be nice and use only x0 in addition to a

        move    x:(r0),a        ;move head pointer into accumulator
        move    x:(r0+1),x0     ;move tail pointer into x0 for CMP

        cmp     x0,a            ;test to see if they are the same
                                ; (this also sets ZF)

        move    ssh,x0          ;pop x0 back off the stack
        POP_ACC A


        rts     ;return from subroutine


```

```
; queue_full
; Functional Specification
;
; Description:      queue_full takes the head and tail pointers, does
; a little bit of modulo arithmetic to increment the tail pointer, and tells
; us whether the queue is full or not.
;
; Operation:        The function gets the head and tail pointers, and increments
; the tail pointer. If the tail pointer points to an invalid slot, it's wrapped
; back to the first slot. After this, the two pointers are compared; if they are
; equal (the queue is full), then the ZF flag is set, otherwise unset.
;
; Arguments:        r0 - address of the queue to look at
;
; Return Values:    ZF - set if queue is full
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  queue_full
queue_full:                     ;assembly entry point

        ;we use the accumulators due to CMP/SUB, so save them:
        PUSH_ACC A
        ;save x0, x1
        move    x0,ssh
        move    x1,ssl          ;and then x1 into ssl

        move    x:(r0),x0       ;move head pointer into x0
        move    x:(r0+1),a      ;move tail pointer into accumulator for possible
                                ; SUB
        move    x:(r0+2),x1     ;move size into x1 so we can compare it

```

```
        add     #>1,a            ;we increment the pointer to the next address

        cmp     x1,a            ;we see if the pointer is too large (x1<a)
        jle     queue_full_nowrap
                                ;if tail >= size, wrap to zero
       ;jgt     queue_full_wrap
queue_full_wrap:
        clr     a               ;we reset our tail pointer to 0 to wrap

queue_full_nowrap:

        cmp     x0,a            ;test to see if they are the same
                                ; (this also sets ZF)

        move    ssl,x1          ;pop x1 back off the stack
        move    ssh,x0          ;restore x0
        POP_ACC A


        rts     ;return from subroutine

; dequeue
; Functional Specification
;
; Description:      dequeue takes a queue and returns the first object
; of the queue. it blocks if there is nothing in the queue to return.
;
; Operation:        Dequeue takes the queue and calls queue_full to see
; if it is full; if so, it waits for it to dequeue. once that is checked,
; it then puts the value of the item in a0, and increments the head
; pointer. if this pointer is too large (>=size) it is reset back to zero,
; to wrap to the first element.
;
; all of this is wrapped in critical code statements because it is
; ill-defined what a queue should do when interrupted in this process.
;
; Arguments:        r0 - the address of the queue to dequeue
;
; Return Values:    a1 - the value of the item at the front of the queue
;                   a0 is cleared, a2 sign-extended as necessary.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
```

```
; Registers changed: a2, a1, a0 (a)
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  dequeue
dequeue:                        ;assembly entry point

        ;we use the accumulators due to CMP/INC, so save them:
        PUSH_ACC b

        ;and now save x0 & n0
        move    x0,ssh
        move    n0,ssl

        ;check if there are no objects to dequeue
dequeue_block_loop:
        jsr     queue_empty
        jeq     dequeue_block_loop      ;if there is nothing to dequeue, block.

        ;since we grab the head pointer, from here on it is critical code
        START_CRITICAL
        ;-- START OF CRITICAL CODE --

        move    x:(r0)+,b       ;grab the head pointer so that we can modify it
                                ;(and do comparisons, increments on it)
        move    b1,n0           ;also put it in n0 so we can grab the item
        move    x:(r0+1),x0     ;grab size for wrap usage

        move    (r0)+           ;increment pointer into data space
        move    (r0)+
        move    x:(r0+n0),a     ;grab our item into a

        ;we take care of wrapping, now

        add     #1,b            ;increment the head pointer
        cmp     x0,b            ;is the head pointer too large?
        jle     dequeue_no_wrap ;do not wrap if the size is larger than the pointer
;       jgt     dequeue_wrap
dequeue_wrap:
        clr     b               ;we clear the head pointer.

dequeue_no_wrap:
        move    (r0)-           ;decrement pointer to start of queue
        move    (r0)-
        move    (r0)-
        move    b1,x:(r0)       ;replace the head pointer

        ; -- END OF CRITICAL CODE --
        END_CRITICAL

```

```
        ;end of body code, now clean up registers
        move    ssl,n0          ;restore n0/x0
        move    ssh,x0

        POP_ACC b

        rts     ;return from subroutine
        ;end of dequeue


; enqeueue
; Functional Specification
;
; Description:      enqueue takes a single argument in a0 to add to the end of
; the queue. it places it at the end of the queue; if there is no space, it
; will block until there is space.
;
; Operation:        enqueue first checks if there is space left in the queue .
; if there is no space, it will block until there is space. it then places the
; item at the end of the queue, and then increments the pointer and wraps it
; as necessary to fit within the data block.
;
; Arguments:        a1 - the data to place at the end of the queue
;                        (we don't use a0/a2)
;                   r0 - the address of the queue to update
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes: much of this code is ill-defined as to what to do when
; interrupted, so we block of interrupts for the update portion.
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  enqueue
```

```
enqueue:                        ;assembly entry point

        ;save all registers we use so we can restore later
        ;we use the accumulators due to CMP/SUB, so save them:
        PUSH_ACC A
        ;then we used n0/x0, so save them:
        move    x0,ssh
        move    n0,ssl


        ;check if there is any space left to queue
enqueue_block_loop:
        jsr     queue_full      ;check if queue is full
        jeq     enqueue_block_loop      ;if so, loop


        ;since we grab the tail pointer, from here on it is critical code
        START_CRITICAL
        ;-- START OF CRITICAL CODE --

        move    x:(r0+1),n0     ;we will use the tail pointer as an offset

        ;we put the object into the queue:
        move    (r0)+           ;increment the pointer to the beginning of data
        move    (r0)+
        move    (r0)+
        move    a1,x:(r0+n0)    ;grab from a0 and put it into the queue
        move    (r0)-           ;and restore r0 to the start of the queue struct
        move    (r0)-
        move    (r0)-

        ;and now we worry about wrapping the tail pointer.

        move    x:(r0+1),a      ;use the a accumulator for tests etc
        add     #>1,a           ;increment the tail pointer
        move    x:(r0+2),x0     ;grab the size into x0 so we can compare with it

        cmp     x0,a            ;compare size and tail pointer
        jle     enqueue_no_wrap ;do not wrap if size>pointer
       ;jgt     enqueue_wrap
enqueue_wrap:
        clr     a               ;reset tail pointer to 0

enqueue_no_wrap:
        move    a1,x:(r0+1)     ;update tail pointer in the queue



        ; -- END OF CRITICAL CODE --
        END_CRITICAL

        ;end of body code, now restore all registers
        move    ssl,n0          ;restore n0/x0
        move    ssh,x0

        POP_ACC A
```

```
        rts     ;return from subroutine
;end function




        endsec
```

8.4 Display (display.inc, display.asm)

display.inc contains important defines and constants for the display code, located in display.asm.

display.asm contains the display code. We assume a HD44780-compatible LCD that is connected to the CPLD and address/data busses as in the LCD schematic.

```
; display.inc
;
; This file offers several key includes for the display functionality,
; such as size of memory buffers and constants, etc.
;
; Revision History:
;
; 2011/2/15     Raymond Jimenez         initial revision
; 2011/6/12     Raymond Jimenez         cleaned up for documentation
;
;


;allocate 256 words for the LCD queue
DISPLAY_QUEUE_SIZE      EQU     $200


;our display is located at 0x100000
DISPLAY_ADDR            EQU     $100000

;allocate 255 characters to be written to the LCD
; at any given time.
DISPLAY_TEMP_BUF        EQU     $ff


;we steal these from interfac.h so that we can
; use them in assembly.

STATUS_PLAY             EQU     0
STATUS_FASTFWD          EQU     1
STATUS_REVERSE          EQU     2
STATUS_IDLE             EQU     3
STATUS_ILLEGAL          EQU     4

TIME_NONE               EQU     65535

;            our display looks like the following:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
;           | artist                 |> Play -127dB
;           | title                      | time    |
;


STATUS_ADDR             EQU     26
ATTEN_ADDR              EQU     34

ATTEN_SHIFT           EQU     3   ;shift over 3 to get full integer dB

```

```
; display.asm
;
; This file contains the hardware-level interface code for the LCD display.
; It offers the functions:
;
; void display_time(unsigned int) - display the passed time on the LCD
; void display_status(unsigned int) - display the passed status on the LCD
; void display_title(char far *) - display the title on the LCD
; void display_artist(char far *) - displays the artist on the LCD
;
; void display_update(void) -- updates the LCD; should be called from a timer
;  interrupt
;
; void display_init(void) - initializes the LCD structures and hardware; should
;  be called before any other display calls are made
;
; Revision History:
;
; 2011/2/15     Raymond Jimenez         initial revision
; 2011/6/12     Raymond Jimenez         cleaned up for documentation
;
;


        section display    ;here's our declaration for our file
 INCLUDE 'display.inc'
 INCLUDE 'macros.inc'

        ;we declare volatile data structures first
        org x:                  ;keep all the large stuff in sram/dram/x ram

        ; VOLATILE DEFINITONS START HERE

display_queue:
        ds $1                   ;keep a single 24-bit counter around for the
                                ; head offset
        ds $1                   ;and the tail offset

        ds DISPLAY_QUEUE_SIZE   ;queue data

        ; we locate several string constants here to simplify our life.
        ; we simply assume all strings passed to our writing functions
        ;  are located in x-space; for C-passed strings this is certainly
        ;  true (many C constants/data structures are located in X space)
        ;  so we adopt the same standard for strings.

        ; our display looks like the following:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
;           | artist                |> Play -127dB
;           | title                     | time    |
;


        ;several fields need to be cleared every time they are used,
        ; like the title and artist fields. we assume a 2x40 LCD.
        ; we have:
```

```

clear_artist_str:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
        dc "                          ",$00
clear_title_str:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
        dc "                               ",$00
clear_time_str:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
        dc                                "        ",$00

clear_vol_str:
;                    1         2         3         4
;           1234567890123456789012345678901234567890
        dc                                   "      ",$00

time_minute_sep:
        dc ":",$00
time_tenths_sep:
        dc ".",$00

atten_prefix:
        dc "-",$00
atten_suffix:
        dc "dB",$00

        ;we have several constants corresponding to what to display
        ; for a given status:
status_play_str:
        dc "|> Play ",$00
status_fastfwd_str:
        dc ">> FF   ",$00
status_reverse_str:
        dc "<< Rev  ",$00
status_idle_str:
        dc "[] Stop ",$00
status_illegal_str:
        dc "OTL     ",$00

        ;we require a temporary buffer to place our results from
        ; dec2string.
convert_temp_buf:
        ds $ff          ; we allocate 255 words, which will inevitably be
                        ;  255 right-aligned ASCII characters.
        dc $00          ; and we null terminate, in case.

        ;and from here, we declare things to be burned in ROM
        org p:

        ; CONSTANT DEFINITONS START HERE

        ;we have a lookup table for all of the statuses:
status_str_table:
        dc status_play_str              ;0 - play
```

```
        dc status_fastfwd_str           ;1 - fastfwd
        dc status_reverse_str           ;2 - reverse
        dc status_idle_str              ;3 - idle
        dc status_illegal_str           ;4 - illegal

        ; PROGRAM CODE STARTS HERE

        ;and finally, program code
        global  display_update
display_update:                 ;assembly entry only
        ; we are here from an interrupt, so be quick and careful
        ; do not worry about clearing the interrupt pin just yet
        move    r0,ssh                  ;put r0 on stack; we can't modify it
        move    r1,ssl

        ;put a on stack, must keep it clean
        PUSH_ACC a

;Save all registers to be used.
        move    #display_queue,r0       ;setup for calling dequeue/queue empty

        ;check if there's any commands to be sent
        jsr     queue_empty
        jeq     display_update_done     ;if empty, don't do anything

display_update_process:
        jsr     dequeue                 ;else, we dequeue the command and write

        move    a1,x:DISPLAY_ADDR      ;and then write to the display



display_update_done:

;Restore all registers used.
        ;restore a accumulator first
        POP_ACC a

        move    ssl,r1
        move    ssh,r0                  ;pop r0 back off of stack


        rts

; Fdisplay_init
; Functional Specification
;
; Description:      Fdisplay_init ( void display_init() ) should be called
; from a C calling function to initialize the display mechanics, such as
; queues etc.
;
; Operation:        Fdisplay_init initializes the display command queue
; using queue_init, but that's it.
;
; Arguments:        None.
;
; Return Values:    None.
```

```
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/17    Raymond Jimenez    first revision
;
        global  Fdisplay_init
Fdisplay_init:                      ;C entry point


        ;standard C-caller prologue
        move #0,n6       ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        move r0,ssh      ;we use r0 only
        move r1,ssl      ;and r1 as well

        move #display_queue,r0
        move #DISPLAY_QUEUE_SIZE,r1
        jsr  queue_init  ;init the display queue

        ;we reset the display
        move #>$1,a  ;we are going to reset the display hardware
        jsr  enqueue     ;and enqueue it for the timer to take care of


        ;however, this takes the display 1.53mS -- super long.
        ; our timer is set to go every 50us, so we program in
        ; 32 NOPs to allow the reset to proceed. (32*50us = 1.6mS)
        move #>$0,a  ;do NOPs for the next instruction.
        do   #32,display_init_loop_end          ;do 32 NOPs.

        nop
        jsr  enqueue
```

```
        nop
        nop

display_init_loop_end:


        ;set our output mode: display on
        move #>$0c,a
        jsr  enqueue


        move #>$38,a  ;and make 2-line mode a thing.
        jsr  enqueue     ;and let the display interrupt take care of it

        ;end of body code, now restore all registers

        move ssl,r1      ;restore r1
        move ssh,r0      ;r0 is now back to initial state

        ;standard C-caller epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.


        rts     ;return from subroutine
;end Fdisplay_init


; display_write_string
; Functional Specification
;
; Description:      This takes an ASCII null-terminated string and
; writes it to the LCD display. The string must be in X space.
;
; Operation:        The function loops over the string, reading each
; character at a time. If the string is a NULL ($0), it returns.
; It takes the lowest byte of the word and enqueues it OR'd with
; 0x000100, which is the R/W bit on the LCD (indicating a character).
;
; Arguments:        r0 - pointer to the string we want to enqueue
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
```

```
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  display_write_string
display_write_string:                       ;assembly entry point

        ;save all registers we use so we can restore later

        move    r0,ssh          ;we save r0 since we'll alter it
        move    r1,ssl          ;as well as r1
        ;we also use a0, so keep it on the stack
        PUSH_ACC a

        move    r0,r1           ;we'll use r1 as the real counter
        move    #display_queue,r0
                                ;and r0 pointing to the queue for
                                ; the enqueue call


        ;initialize a with the first byte:
        clr     a
        move    x:(r1)+,a1

        ;start the main loop:

write_string_loop:
        nop

        and     #>$7f,a         ;make sure we have only ASCII chars
        tst     a       ;see if a is all zeros -> NULL
        jeq     write_string_loop_end ;if so, get out of here

        or      #>$100,a        ;place in the 'R/W' bit on the LCD
        jsr     enqueue         ;now enqueue it to be written

        move    x:(r1)+,a1
        nop
        nop
        nop
        jmp     write_string_loop
write_string_loop_end:


        nop
        nop
        nop
```

```
        POP_ACC a
        move    ssl,r1          ;restore r1
        move    ssh,r0          ;restore r0


        ;end of body code, now restore all registers
        rts     ;return from subroutine
;end display_write_string


; display_set_loc
;
; Functional Specification
;
; Description:      display_set_loc allows one to control the location of
; where a string will be blit onto the LCD display. For simplicity, it takes
; a DD RAM address corresponding to a physical address on our 2x40 LCD.
;
; Row 1: $0-$27
; Row 2: $40-67
;
; There is no error handling; if given an invalid address an unpredictable
; result will occur.
;
; Operation:        display_set_loc enqueues the appropriate command bits
; to set the LCD to the given address, and then relies on the interrupt/timing
; routines to send the command to the LCD.
;
; Arguments:        a1 - DDRAM addresss to position the cursor at
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    Based on the MTC-S40200XRGHS datasheet.
```

```
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  display_set_loc
display_set_loc:                    ;assembly entry point

        ;save all registers we use so we can restore later
        move    r0,ssh              ;we use r0 of course.
        ;and we change the accumulator
        PUSH_ACC a

        ;alright, we alter a to include our nice bit-mask
        ; that turns an address into an LCD command:
        and     #>$7f,a         ;clear out anything above DB6
        or      #>$80,a         ;DB7 needs to be high, nothing else

        move    #display_queue,r0 ;get the address for enqueue
        jsr     enqueue         ;enqueue the command

        ;the LCD datasheet lies and needs a single nop after the
        ; set location for timing
        clr     a
        jsr     enqueue

        ;end of body code, now restore all registers
        POP_ACC a
        move    ssh,r0

        rts     ;return from subroutine
;end display_set_loc


; Fdisplay_status
; Functional Specification
;
; Description:      Fdisplay_status takes a status number and writes the
; appropriate status to the display.
;
; Operation:        Fdisplay_status uses the status number as an offset
; to lookup the appropriate string pointer in x-space, then calls
; display_set_loc to set the location of the string, followed by a
; call to display_write_string to actually blit the string to the display.
;
; Arguments:        a1 - status number to display, per interfac.h
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
```

```
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fdisplay_status
Fdisplay_status:                          ;C entry point

        ;standard C prologue
        move #0,n6       ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl
        move    n1,ssh

        ;we also change b, so preserve it
        PUSH_ACC b

        ;we change the address to display in the upper top right corner, always.
        ; since our strings are fixed length, we can just rely on them to
        ; mask the previous status' text.

        move    a,b     ;keep our status around
        move    #>STATUS_ADDR,a  ;we like to be in the top right corner.
        jsr     display_set_loc
        move    b,a

        ;our argument should be in a1 because we only take a single argument.


        ;as a result, let's move it into n1 and go:
        move    #status_str_table,r1    ;our base table for lookup is
                                        ; status_str_table
        move    a1,n1                   ;move our status into n1 for indexing
        move    p:(r1+n1),r0            ;and then we get the address to pass
                                        ; to our string-display func

        ;we now display the text
        jsr     display_write_string

        ;end of body code, now restore all registers
        POP_ACC b

```

```
        move   ssh,n1
        move   ssl,r1
        move   ssh,r0

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        rts     ;return from subroutine
;end Fdisplay_status


; Fdisplay_title
; Functional Specification
;
; Description:      This function displays the title on the LCD display.
;
; Operation:        Calls display_set_loc to set the display location,
; then clears the location, and then proceeds to write the string to the
; location.
;
; Arguments:
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fdisplay_title
Fdisplay_title:                         ;C entry point

        ;standard C prologue
        move #0,n6       ; k is the amount of local space needed.
```

```
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl

        ;we also change b, so preserve it
        PUSH_ACC a


        ;we first clear the title field in the 2nd row
        move    a1,r1     ;keep our string pointer around
        move    #>$40,a   ;we like to be in the 2nd row.
        jsr     display_set_loc


        ;and then clear:
        move    #clear_title_str,r0
        jsr     display_write_string


        ;now write our real title
        ;(go back to the initial position)
        jsr     display_set_loc

        ;our string to write has to be in x-space, since the title is volatile
        move    r1,r0            ;move our string pointer back in place
        jsr     display_write_string


        ;end of body code, now restore all registers
        POP_ACC a

        move   ssl,r1
        move   ssh,r0

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        rts     ;return from subroutine
;end Fdisplay_title

; Fdisplay_artist
; Functional Specification
;
; Description:      This function displays the artist on the LCD display.
;
; Operation:        Calls display_set_loc to set the display location,
; then clears the location, and then proceeds to write the string to the
; location.
;
; Arguments:        a1 - pointer to ASCII NULL-term'd string to write
;
; Return Values:    None.
```

```
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          Artist name on the display.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fdisplay_artist
Fdisplay_artist:                         ;C entry point

        ;standard C prologue
        move #0,n6       ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl

        ;we also change b, so preserve it
        PUSH_ACC a

        ;we write to the top left of the display:

        move    a1,r1    ;keep our string pointer around
        move    #>$0,a   ;we like to be in the top left corner.
        jsr     display_set_loc

        move    #clear_artist_str,r0 ;now we clear
        jsr     display_write_string

        jsr     display_set_loc      ;go back to top left corner

        ;our string to write has to be in x-space, since the artist is volatile
        move    r1,r0     ;write our real artist
        jsr     display_write_string

```

```

        ;end of body code, now restore all registers
        POP_ACC a

        move   ssl,r1
        move   ssh,r0

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        rts     ;return from subroutine
;end Fdisplay_artist



; Fdisplay_time
; Functional Specification
;
; Description:      This function displays the time on the LCD display.
;
; Operation:        This function divides the time given by 600 to get
; minutes, then calls dec2string, and displays it at the appropriate
; spot. This step is repeated for the next 2 subdivisions: seconds and
; tenths of seconds, each division taking the remainder of the last.
;
; Arguments:        a1 - integer of 10ths of seconds in the song.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: convert_temp_buf - temporary buffer for conversion use
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          Time on the screen.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
```

```
        global  Fdisplay_time
Fdisplay_time:                          ;C entry point

        nop
        nop
        rts

        ;standard C prologue
        move #0,n6      ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        PUSH_ACC a
        PUSH_ACC b
        move r0,ssh
        move x0,ssl


        move    a,b       ;keep our real time around
        move    #>$68-9,a ;we like to be in the bottom right corner.
                          ;(we can write $28 characters to a line,
                          ; so subtract nine to be right-justified)
        ;we clear out the time to get rid of prev. digits
        jsr  display_set_loc
        move #clear_time_str,r0
        jsr  display_write_string

        ;we compare if the time is equal to TIME_NOTIME
        move    b,a

        cmp     #>TIME_NONE,a
        jeq     display_time_done       ;if it is equal, do not display anything

        ;now that we're cleared, let's write the minutes:
        ; (we rely on the builtin C functions to make our life easier)
        move    #600,x0                 ;we divide by 600 tenths of a second ==
                                        ; one minute
        jsr     idiv_x0a                ; a = floor(a/x0)

        ;now we convert our minutes to seconds and display:
        move #convert_temp_buf,r0
        jsr  dec2string                 ;convert a to a string

        move    #>$68-9,a ;we like to be in the bottom right corner.
                          ;(we can write $28 characters to a line,
                          ; so subtract nine to be right-justified)
        jsr     display_set_loc
        jsr     display_write_string    ;write minute string

        ;write the separator between min/sec:
        move    #time_minute_sep,r0
        jsr     display_write_string

        ;we now get the number of seconds:
        ; seconds = (tenths of a second % 600) / 10
        move    b,a     ;retrieve our whole value
```

```
        move    #>600,x0
        jsr     imod_x0a ; a = a%600

        ;we divide by ten to get seconds:
        move    #>10,x0
        jsr     idiv_x0a ; a = floor(a/10)

        ;and we turn this into a string and blit it:
        move    #convert_temp_buf,r0
        jsr     dec2string
        jsr     display_write_string

        ;we write the divider between tenths and seconds:
        move    #time_tenths_sep,r0
        jsr     display_write_string

        ;we now have to deal with tenths, which is easy:
        ; tenths = tenths % 10
        move    b,a     ;grab our original value again
        move    #>10,x0 ;mod 10
        jsr     imod_x0a ; a = a%10

        ;turn this into a string and blit it to the display:
        move    #convert_temp_buf,r0
        jsr     dec2string
        jsr     display_write_string

        ;and that's it, for displaying the time!




display_time_done:

        ;end of body code, now restore all registers
        move ssl,x0
        move ssh,r0
        POP_ACC b
        POP_ACC a

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        rts     ;return from subroutine
;end Fdisplay_time

; Fdisplay_volume
; Functional Specification
;
; Description:      This function displays the current volume on the screen.
;
; Operation:        It calls get_atten to display the current volume, then
; shifts this right 3 times (divide by 8) to get the number of dB attenuated.
; This is passed to dec2string, which we write to the display, followed by
; "dB".
```

```
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: convert_temp_buf - temporary buffer for binary->ASCII
;                                       conversion
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          Volume on the screen, in units of dB.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fdisplay_volume
Fdisplay_volume                        ;C entry point

        ;standard C prologue
        move #0,n6        ; k is the amount of local space needed.
        move ssh,x:(r6)+  ; save the return address.
        move (r6)+n6      ; allocate local stack space of size k.

        ;save all registers we use so we can restore later
        PUSH_ACC a
        move r0,ssh


        ;we first clear out our portion of the display

        move    #>ATTEN_ADDR,a ;we like to be located after the status
        jsr  display_set_loc
        move #clear_vol_str,r0
        jsr  display_write_string

        ;now, write our "-" sign
        jsr  display_set_loc
        move #atten_prefix,r0
        jsr  display_write_string

```

```
        jsr  get_atten   ;we now work with the attenuation
        asr  #ATTEN_SHIFT,a,a ;shift over to get integer dB

        ;and we convert to a number and write it to the screen
        move    #convert_temp_buf,r0
        jsr     dec2string
        jsr     display_write_string

        ;this is followed by writing the " dB"

        move #atten_suffix,r0
        jsr  display_write_string



        ;end of body code, now restore all registers
        move    ssh,r0
        POP_ACC a

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        rts     ;return from subroutine
;end Fdisplay_volume


        endsec          ;end display

```

8.5 Keypad (keyfunc.inc, keyfunc.asm)

keyfunc.inc contains important definitions for the keypad code, mostly keyscan code definitions.

keyfunc.asm contains the keypad processing code and the event handler installed to handle the keypad IRQ.

```
; keyfunc.inc
;
; This file contains several important definitions that the keypad code
; depends on, such as the definitions for the keycodes.
;
; Revision History
;
; 2011/2/16     Raymond Jimenez         initial revision
; 2011/6/12     Raymond Jimenez         cleaned up for documentation
;


;if our translated key value is equal to this, don't
; do anything
KEY_NOP         EQU     $ffffff

;we steal the definitions below from interfac.h in order
; to use them in our assembly:

KEY_TRACKUP     EQU     0
KEY_TRACKDOWN   EQU     1
KEY_PLAY        EQU     2
KEY_RPTPLAY     EQU     3
KEY_FASTFWD     EQU     4
KEY_REVERSE     EQU     5
KEY_STOP        EQU     6
KEY_VOLUP       EQU     7
KEY_VOLDOWN     EQU     8
KEY_ILLEGAL     EQU     9

HAVE_KEY        EQU     1
NO_KEY          EQU     0

KEYPAD_ADDR     EQU     $101000

KEY_VALID_MASK  EQU     $00000f
```

```
; keyfunc.asm
;
; This file contains procedures related to the keypad, such as getkey and the
; keypad interrupt handler.
;
; keypad_handler - the event handler called when a new key has been pressed
; Fkey_init      - initialization code for the keypad; call before doing
;                  anything related to the keypad (including installing event
;                  handlers)
; Fkey_available - call to see if there is a new key available for processing
; Fgetkey        - call to get a key; blocks on key input
;
; Revision History
;
; 2011/2/16     Raymond Jimenez         initial revision
; 2011/6/12     Raymond Jimenez         cleaned up for documentation
;



        section keyfunc         ;here's our declaration for our file
 INCLUDE 'keyfunc.inc'
 INCLUDE 'macros.inc'

        ;we declare volatile data structures first
        org x:                  ;keep all the volatile variables in sram/dram

        ; VOLATILE DEFINITONS START HERE

raw_key:
        ds      $1      ;we have a word to contain the raw keycode

raw_havekey:
        ds      $1      ;and one word as a boolean to see if we have a new key

        ; CONSTANT DEFINITIONS START HERE
        org p:          ;keep all constants in P-space for burning

        ;we now define a translation table for our keypad:
        ;
        ;our keypad looks sort of like this:
        ;
        ; +----+----+----+----+
        ; | 0  | 1  | 2  | 3  |
        ; +----+----+----+----+
        ; | 4  | 5  | 6  |    |
        ; +----+----+----+----+
        ; | 8  | 9  | 10 | 7  |
        ; +----+----+----+----+
        ; | 12 | 13 | 14 |    |
        ; +----+----+----+ 15 +
        ; |    | 11 |    |    |
        ; +----+----+----+----+
        ;
        ; where we have 15 keys in a not-quite-grid-like fashion.
        ; (numbers are values returned when reading from the keypad)

```

```

key_trans_table:
        dc      KEY_NOP         ;key 0 - don't do anything
        dc      KEY_NOP         ;key 1 - don't do anything
        dc      KEY_NOP         ;key 2 - don't do anything
        dc      KEY_NOP         ;key 3 - don't do anything
        dc      KEY_NOP         ;key 4 - don't do anything
        dc      KEY_TRACKUP     ;key 5 - track up
        dc      KEY_VOLUP       ;key 6 - increase volume
        dc      KEY_RPTPLAY     ;key 7 - repeat play (loop)
        dc      KEY_REVERSE     ;key 8 - reverse
        dc      KEY_STOP        ;key 9 - stop
        dc      KEY_FASTFWD     ;key 10 - fast forward
        dc      KEY_NOP         ;key 11 - don't do anything
        dc      KEY_NOP         ;key 12 - don't do anything
        dc      KEY_TRACKDOWN   ;key 13 - track down
        dc      KEY_VOLDOWN     ;key 14 - decrease volume
        dc      KEY_PLAY        ;key 15 - play
        ;we should never get higher values of keys, otherwise
        ; something is wrong.
        dc      KEY_NOP         ;key 16 - don't do anything, but it's
                                ; odd that it's even here...
        dc      KEY_NOP         ;key 17 - don't do anything
        dc      KEY_NOP         ;key 18 - don't do anything
        dc      KEY_NOP         ;key 19 - don't do anything
        dc      KEY_NOP         ;key 20 - don't do anything


        ; PROGRAM CODE STARTS HERE

        ;and finally, program code

; keypad_handler
; Functional Specification
;
; Description:      keypad_handler is the interrupt handler for the keypad.
; It sets up raw_key and raw_havekey so that the other key functions work.
;
; Operation:        Executes a move instruction to move the raw
; keycode into the variable, and then moves HAVEKEY into raw_havekey
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
```

```
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  keypad_handler
keypad_handler:                         ;assembly entry point
        ;remember to save all registers changed! interrupt handler, so very
        ; important!
        move    x0,ssh

        move    x:KEYPAD_ADDR,x0        ;move our key into a temp buffer
                                        ; (this also clears the interrupt by
                                        ; reading from the keypad chip)
        move    x0,x:raw_key            ; then into our variable

        move    #HAVE_KEY,x0            ;and then signal we have a key
        move    x0,x:raw_havekey

        ;restore registers
        move    ssh,x0

        rti     ;interrupt handlers return w/ rti
;end keypad_handler

; Fkey_init
; Functional Specification
;
; Description:      Fkey_init should be called before any other key functions
; are called in order to initialize the data structures.
;
; Operation:        Fkey_init clears out raw_havekey to zero.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
```

```
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: a.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fkey_init
Fkey_init:                  ;C entry point

        ;standard C prologue
        move #0,n6      ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6    ; allocate local stack space of size k.

        ;save all of the registers we use
        move    r0,ssh   ;save r0

        move    #NO_KEY,r0
        move    r0,x:raw_havekey

        ;restore registers
        move    ssh,r0

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        ;save all registers we use so we can restore later

        rts     ;return from subroutine
;end Fkey_init

; Fkey_available
; Functional Specification
;
; Description:      Fkey_available returns 0 if there is no key, otherwise
; non-zero.
;
; Operation:        Fkey_available checks if raw_havekey has been set. If so,
; it checks the value of raw_key to see if it is a valid key; if not, it clears
; raw_havekey and returns. If it is a valid key, it returns non-zero.
;
; Arguments:        None.
;
; Return Values:    a1 - non-zero if there is a valid key waiting
;
; Global Variables: None.
```

```
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: a.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fkey_available
Fkey_available:                     ;C entry point

        ;standard C prologue
        move #0,n6       ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all of the registers we use
        move    r0,ssh   ;save both r0/n0
        move    n0,ssl

        move    x:raw_havekey,a         ;we check if we have a key
        tst     a                       ;do we have a key?
        jeq     key_avail_done          ;if not, we're done


        move    #key_trans_table,r0     ;start the lookup
        move    x:raw_key,a             ;move our key into a so we can clear
        and     #>KEY_VALID_MASK,a      ;clear extreneous high bits
        move    a1,n0                   ;we use the key as an index

        move    p:(r0+n0),a             ;grab the keycode into a
        eor     #KEY_NOP,a              ;XOR it with the NOP command
                                        ; -> will return zero only if NOP
                                        ; otherwise, guaranteed to be non-zero

key_avail_done:

        ;end of body code, now restore all registers
        move    ssl,n0
        move    ssh,r0
```

```

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        ;save all registers we use so we can restore later

        rts     ;return from subroutine
;end function


; Fgetkey
; Functional Specification
;
; Description:      Fgetkey returns the translated keycode of a key; it blocks
;if there is no key available.
;
; Operation:        Fgetkey calls Fkey_available to see if there is a key ready,
; if not it loops until there is. It then grabs the value of raw_key, clears
; unnecessary bits, and looks up that value in a table, which is then put into
; a1.
;
; Arguments:        None.
;
; Return Values:    a1 - translated keycode
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: a.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fgetkey
Fgetkey:                    ;C entry point

```

```
        ;standard C prologue
        move #0,n6       ; k is the amount of local space needed.
        move ssh,x:(r6)+ ; save the return address.
        move (r6)+n6     ; allocate local stack space of size k.

        ;save all of the registers we use
        move    r0,ssh   ;save both r0/n0
        move    n0,ssl

getkey_block_loop:
        jsr     Fkey_available          ;check if there are keys available
        jeq     getkey_block_loop       ;if not, keep on looping

        move    #key_trans_table,r0     ;put the lookup table in a base pointer
        move    x:raw_key,a             ;move our key into a so we can clear
        and     #>KEY_VALID_MASK,a      ;clear extreneous high bits
        move    a1,n0                   ;we use the key as an index

        move    p:(r0+n0),a             ;grab the keycode into a, we're done

        move    #NO_KEY,r0
        move    r0,x:raw_havekey   ;now we no longer have a key

        ;end of body code, now restore all registers
        move    ssl,n0
        move    ssh,r0

        ;standard C epilogue
        move #0+1,n6
        move (r6)-n6    ; deallocate local stack space, set ccr flags.
        tst  a
        move x:(r6),ssh ; get the return address.
        ;save all registers we use so we can restore later

        rts     ;return from subroutine
;end Fgetkey




        endsec          ;end keypad

```

8.6 IDE Interface

ide.inc contains important definitions for the IDE code, mostly IDE standard definitions.

ide.asm contains the IDE block reading code, as well as the initialization code.

```

;the various IDE register addresses
; some are read only, others write only,
; most r/w. please check the ATA spec for details.

;the addresses come from how we've wired up the
; bus to the IDE connector.


IDE_DATA_REG            EQU     $103000
IDE_FEATURE_REG         EQU     $103100
IDE_ERROR_REG           EQU     $103100
IDE_SEC_CT_REG          EQU     $103200
IDE_SEC_NUM_REG         EQU     $103300
IDE_CYL_LOW_REG         EQU     $103400
IDE_CYL_HIGH_REG        EQU     $103500
IDE_DEV_HEAD_REG        EQU     $103600
IDE_STATUS_REG          EQU     $103700
IDE_COMMAND_REG         EQU     $103700

IDE_ALT_STATUS_REG      EQU     $102e00
IDE_ALT_CONTROL_REG     EQU     $102e00

;all registers are 8 bits long
IDE_REG_LENGTH          EQU     8


;now we define various commands

;to enable LBA and device 0, write to dev/head reg:
IDE_SELECT_0_LBA        EQU     $40

;to read from the device:
IDE_READ_SEC            EQU     $20

;to identify:
IDE_IDENTIFY            EQU     $ec

;to set faster speeds
IDE_SET_FEATURES        EQU     $ef

IDE_SET_TRANSFER_MODE   EQU     $03
IDE_SET_FASTER_PIO      EQU     $0a ;support PIO-2, no need for IORDY

;and now, various bits within the status buffer:
IDE_STAT_DRDY           EQU     6
IDE_STAT_BSY            EQU     7
IDE_STAT_DRQ            EQU     3
IDE_STAT_ERR            EQU     0

;an IDE block is always 256 words:
IDE_BLOCK_SIZE          EQU     256

IDE_VALID_DATA          EQU     $ffff
```

```

; ide.asm
;
; This file offers several IDE routines, such as setup and other things.
;

        section ide
 INCLUDE 'macros.inc'
 INCLUDE 'ide.inc'
        org     x:

;no variables yet, i think

        org     p:              ;we want to be located in program space

; Fsetup_ide
; Functional Specification
;
; Description:      Fsetup_ide sets up the IDE drive by selecting device 0 and
; setting it to LBA mode. We assume that a drive is attached and in the master
; position.
;
; Operation:        It writes IDE_SELECT_0_LBA to address IDE_DEVHEAD_REG, which
; is the device/head register. It then loops until the drive is ready for
; activity that is, until DRDY is set.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/03/08    Raymond Jimenez    first revision
;
```

```
        global  Fsetup_ide
Fsetup_ide:                 ;C entry point
        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

        ;save all registers we use so we can restore later
        move    r0,ssh

        ;wait until the drive is ready for selecting
        jsr     wait_for_ide

        ;select the device
        move    #>IDE_SELECT_0_LBA,r0
        move    r0,x:IDE_DEV_HEAD_REG

        move    #>IDE_IDENTIFY,r0
        move    r0,x:IDE_COMMAND_REG

        ;wait until it has reported back
        jsr     wait_for_ide

        ;we now set a faster transfer mode
        move    #>IDE_SET_TRANSFER_MODE,r0
        move    r0,x:IDE_FEATURE_REG

        move    #>IDE_SET_FASTER_PIO,r0
        move    r0,x:IDE_SEC_CT_REG

        move    #>IDE_SET_FEATURES,r0
        move    r0,x:IDE_COMMAND_REG

        ;wait until it has reported back
        jsr     wait_for_ide

        ;restore registers
        move    ssh,r0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts     ;return from subroutine
;end Fsetup_ide


; wait_for_ide
; Functional Specification
;
; Description:      wait_for_ide simply reads the status register and waits
; until BSY and DRDY are cleared and set, respectively.
;
; Operation:        reads from IDE_STATUS_REG, checks BSY and DRDY bits,
; loops if not ready
```

```
;
; Arguments:        None
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/03/08    Raymond Jimenez    first revision
;
        global  wait_for_ide
wait_for_ide:                       ;assembly entry point

        ;save all registers we use so we can restore later
        move    r0,ssh

wait_for_ide_loop:
        move    x:IDE_STATUS_REG,r0
        nop     ;GDB delay
        jset    #IDE_STAT_BSY,r0,wait_for_ide_loop

        ;end of body code, now restore all registers
        move    ssh,r0
        nop
        rts     ;return from subroutine
;end wait_for_ide


; Fget_blocks
; Functional Specification
;
; Description:      Fget_blocks grabs a number of blocks from the specified
; address on the IDE disk.
;
; Operation:        It first sets up the head registers, and then commits the
; command to read from the disk. We don't actually detect errors; we depend on
```

```
; the IDE drive to stop giving us data, resulting in returning less blocks
; than the calling function expects.
;
; Arguments:        first (a) - block number to start reading from
;                   second (b) - number of blocks to read
;                   third (r6-1) - (on stack) - pointer of address to write to
;
; Return Values:    a - number of blocks read
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: a
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/03/08    Raymond Jimenez    first revision
;
        global  Fget_blocks
Fget_blocks:                ;C entry point
        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl
        PUSH_ACC b

        move    x:(r6-2),r1    ;grab our pointer (arg 3) off the stack

        ;note that a carries an unsigned long int, which is 2 words.
        ; as a result, we have MSW in a1, LSW in a0.

        ;setup the registers for the read
        move    a0,x:IDE_SEC_NUM_REG
        move    b1,x:IDE_SEC_CT_REG

        ;since we only have a 24-bit counter for blocks, assume [15:8] will
```

```
        ; go to IDE_CYL_LOW
        asr     #IDE_REG_LENGTH,a,a
        move    a0,x:IDE_CYL_LOW_REG

        ;and then [23:16] go to IDE_CYL_HIGH
        asr     #IDE_REG_LENGTH,a,a
        move    a0,x:IDE_CYL_HIGH_REG

        ;and then we need to put [27:24] in the last 4 bits of
        ; IDE_DEV_HEAD_REG
        asr     #IDE_REG_LENGTH,a,a
        move    a0,a1
        and     #>$ff,a
        or      #>IDE_SELECT_0_LBA,a
        move    a1,x:IDE_DEV_HEAD_REG

        ;now execute the command
        move    #>IDE_READ_SEC,r0
        move    r0,x:IDE_COMMAND_REG

        ;wait until it has reported back
        jsr     wait_for_ide


        ;we now clear a and use it as our data word counter
        clr     a

        ;we can get a maximum of b1 words, so we start looping and comparing
get_all_blocks:
        nop

        ;we can get IDE_BLOCK_SIZE words at least, so limit it to that
        ; we are guaranteed at least one block per ATA spec.
        do #IDE_BLOCK_SIZE,get_block_loop_end
get_block_loop:
        nop

        move    x:IDE_DATA_REG,b
        and     #>IDE_VALID_DATA,b
        move    b1,x:(r1)+

        jsr     wait_for_ide

        ;spacing nop
        nop
        nop
        nop
        nop
        nop
        nop
get_block_loop_end:


        add     #>1,a

        ;and then see if we have more data to run:
        move    x:IDE_STATUS_REG,b      ;check the status reg
```

```
        and     #>(1<<IDE_STAT_DRQ),b      ;check the DRQ bit
        jeq     get_all_blocks_end        ;if equal, get out of here, no more
                                          ; blocks to read
        ;spacing nop
        nop
        nop
        nop
        jmp     get_all_blocks

get_all_blocks_end:


        ;restore registers
        POP_ACC b
        move    ssl,r1
        move    ssh,r0


        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts     ;return from subroutine
;end Fget_blocks

        endsec
```

8.7 Sound (DAC & MP3 Decoder)

sound.inc contains most of the register definitions and command definitions for both the MP3 decoder (VS1053b) and the DAC (WM8741).

sound.asm contains the DMA handling code, the interrupt handling code, and the interface code for the MP3 player. We assume everything is hooked up as in the schematic.

```


; the following are helpful constants when dealing with ESSI0, the
; VLSI VS1053b MP3 decoder chip. data and control are both run over
; ESSI0.

; START DECODER SECTION

;the ESSI ports expect left aligned data, so we shift over this number of bits:
ESSI_BYTE_SHIFT EQU     16
;we deal with 16-bit words in SCI mode
ESSI_SCI_SHIFT  EQU     8


;and a mask just to mask off a single byte, for macro convenience
BYTE_MASK       EQU     $f


;initial ESSI0_CRA, set to 1.6MHz per DAC startup. once we change clock freq up,
; we can do higher serial too.
; we also start with 16 bit words since we're writing to SCI initially.

ESSI0_CRA_1ST   EQU     $100831         ; 0001 0000 0000 1000 0011 0010
                                        ; 0--- --0- ---- -000 ---- ---- reserved
                                        ; -0-- ---- ---- ---- ---- ---- SC1 as flag
                                        ; --01 0--- ---- ---- ---- ---- 16b words
                                        ; ---- -0-- ---- ---- ---- ---- align left
                                        ; ---- ---0 0000 ---- ---- ---- sequential transmit mode
                                        ; ---- ---- ---- 1--- ---- ---- no prescaler
                                        ; ---- ---- ---- ---- 0011 0010 divide by 49+1
                                        ;                               (80/50 = 1.6MHz)

;for initialization we don't depend on interrupts, instead doing polling.

ESSI0_CRB_1ST   EQU     $001d3c         ; 0000 0000 0001 1101 0011 1100
                                        ; 0--- ---- ---- ---- ---- ---- no RX ex. interrupt
                                        ; -0-- ---- ---- ---- ---- ---- no TX ex. interrupt
                                        ; --0- ---- ---- ---- ---- ---- no RX slot interrupt
                                        ; ---0 ---- ---- ---- ---- ---- no TX slot interrupt
                                        ; ---- 0--- ---- ---- ---- ---- no RX interrupt
                                        ; ---- -0-- ---- ---- ---- ---- no TX interrupt
                                        ; ---- --0- ---- ---- ---- ---- RX enabled (will do later)
                                        ; ---- ---0 ---- ---- ---- ---- TX0
```

```
                                        (STD) enabled (will do later)
                                        ; ---- ---- 0--- ---- ---- ---- TX1 (SC0) disabled
                                        ; ---- ---- -0-- ---- ---- ---- TX2 (SC1) enabled (will do later)
                                        ; ---- ---- --0- ---- ---- ---- normal mode
                                        ; ---- ---- ---1 ---- ---- ---- synchronous mode
                                        ; ---- ---- ---- 1--- ---- ---- clock out data on falling edge
                                        ; ---- ---- ---- -1-- ---- ---- negative edge triggers word (positive is end of word)
                                        ; ---- ---- ---- --0- ---- ---- frame sync w/ first bit
                                        ; ---- ---- ---- ---1 0--- ---- bit sync for SCI, need to transmit consecutively
                                        ; ---- ---- ---- ---- -0-- ---- MSB shifted first
                                        ; ---- ---- ---- ---- --1- ---- internal clock
                                        ; ---- ---- ---- ---- ---1 ---- SC2 is an output
                                        ; ---- ---- ---- ---- ---- 1--- SC1 is an output
                                        ; ---- ---- ---- ---- ---- -1-- SC0 is an output
                                        ; ---- ---- ---- ---- ---- --00 don't care, don't use flags

;normal mode, we run at decoder 55MHz, so since max serial is CLKI/7, we get to
; run serial at 6.66MHz now
ESSI0_CRA_SCI   EQU     (((ESSI0_CRA_1ST&(~M_PM)))|$05)
ESSI0_CRB_SCI   EQU     ESSI0_CRB_1ST

;we switch modes between SCI and SDI - for data, we use SC2 as a chip select,
; auto-selecting SDI whenever we move in a byte of data.
;
; (this is the only reason we do SDI - so we can enable DMA and write to our
; decoder quite fast)


ESSI0_CRA_DATA  EQU     ((ESSI0_CRA_1ST&(~M_PM))|$05)
ESSI0_CRB_DATA  EQU     $003c3c         ; 0000 0000 0011 1100 0011 0000
                                        ; 0--- ---- ---- ---- ---- ---- no RX ex. interrupt
                                        ; -0-- ---- ---- ---- ---- ---- no TX ex. interrupt
                                        ; --0- ---- ---- ---- ---- ---- no RX slot interrupt
                                        ; ---0 ---- ---- ---- ---- ---- no TX slot interrupt
                                        ; ---- 0--- ---- ---- ---- ---- no RX interrupt
                                        ; ---- -0-- ---- ---- ---- ---- no TX interrupt
```

```
                                        ; ---- --0- ---- ---- ---- ---- RX disabled
                                        ; ---- ---0 ---- ---- ---- ---- TX0 (STD) enabled (will do later)
                                        ; ---- ---- 0--- ---- ---- ---- TX1 (SC0) disabled
                                        ; ---- ---- -0-- ---- ---- ---- TX2 (SC1) disabled
                                        ; ---- ---- --1- ---- ---- ---- network mode (on-demand mode)
                                        ; ---- ---- ---1 ---- ---- ---- synchronous mode
                                        ; ---- ---- ---- 1--- ---- ---- clock out data on falling edge
                                        ; ---- ---- ---- -1-- ---- ---- negative edge triggers word (positive is end of word)
                                        ; ---- ---- ---- --0- ---- ---- frame sync w/ first bit
                                        ; ---- ---- ---- ---0 0--- ---- word sync for data (XDCS asserted byte-wise)
                                        ; ---- ---- ---- ---- -0-- ---- MSB shifted first
                                        ; ---- ---- ---- ---- --1- ---- internal clock
                                        ; ---- ---- ---- ---- ---1 ---- SC2 is an output
                                        ; ---- ---- ---- ---- ---- 0--- SC1 is an output
                                        ; ---- ---- ---- ---- ---- -0-- SC0 is an output


ESSI0_PCRC_SCI   EQU     $00003a        ; 0000 0000 0000 0000 0011 1010
                                        ; 0000 0000 0000 0000 00-- ---- reserved
                                        ; ---- ---- ---- ---- --1- ---- TX enabled
                                        ; ---- ---- ---- ---- ---1 ---- RX enabled
                                        ; ---- ---- ---- ---- ---- 1--- CLK enabled
                                        ; ---- ---- ---- ---- ---- -0-- SC2 disabled
                                        ; ---- ---- ---- ---- ---- --1- SC1 enabled
                                        ; ---- ---- ---- ---- ---- ---0 SC0 disabled

ESSI0_PCRC_DATA  EQU     $00002c        ; 0000 0000 0000 0000 0010 1101
                                        ; 0000 0000 0000 0000 00-- ---- reserved
                                        ; ---- ---- ---- ---- --1- ---- TX enabled
                                        ; ---- ---- ---- ---- ---0 ---- RX enabled
                                        ; ---- ---- ---- ---- ---- 1--- CLK enabled
                                        ; ---- ---- ---- ---- ---- -1-- SC2
```

```
                                        enabled
                                        ; ---- ---- ---- ---- ---- --0- SC1 enabled
                                        ; ---- ---- ---- ---- ---- ---0 SC0 disabled

;see errata for mask 4H80G, we find that GPIO pins need to be programmed to output
; anyway. this is probably why our shit's not working OTL.

ESSI0_PRRC      EQU     $00003f         ; 0000 0000 0000 0000 0010 1111
                                        ; 0000 0000 0000 0000 00-- ---- reserved
                                        ; ---- ---- ---- ---- --10 1111 all pins output (except for RX)
ESSI0_PDRC      EQU     $00003f         ;set all outputs high if disabled

;we now define some commands/registers for the VS1053 chip:

XDCS_HIGH       EQU     ($ffff<<ESSI_SCI_SHIFT)
XDCS_LOW        EQU     ($0)

XCS_HIGH       EQU      ($ffff<<ESSI_SCI_SHIFT)
XCS_LOW        EQU      ($0)


;please note that all data is shifted over 16 bits as ESSI expects
; left-aligned data.

;write/read commands, shifted over another 8 since they're the first 8 bits
SCI_WRITE_CMD   EQU     ($02<<ESSI_BYTE_SHIFT)
SCI_READ_CMD    EQU     ($03<<ESSI_BYTE_SHIFT)

;register addresses
SCI_MODE        EQU     ($00<<ESSI_SCI_SHIFT)
SCI_CLOCKF      EQU     ($03<<ESSI_SCI_SHIFT)
SCI_VOL         EQU     ($0B<<ESSI_SCI_SHIFT)
SCI_WRAM        EQU     ($06<<ESSI_SCI_SHIFT)
SCI_WRAMADDR    EQU     ($07<<ESSI_SCI_SHIFT)
SCI_AIADDR      EQU     ($0a<<ESSI_SCI_SHIFT)

;register wait times, measured in 100's of ns
SET_CLOCK_WAIT  EQU     977             ;worst case scenario: 1200 XTALI, 12.288MHz crystal

;for other wait times, we assume MP3 decoder has been set to 50MHz for CLKI
SET_VOL_WAIT    EQU     16              ;80 CLKI
SET_WRAM_WAIT   EQU     20              ;100 CLKI
SET_WRAMADDR_WAIT EQU   20              ;100 CLKI
SET_MODE_WAIT   EQU     20              ;100 CLKI
SET_AIADDR_WAIT EQU     42              ;210 CLKI


;we left-align all of our data words to be written
SCI_CLOCKF_VALUE EQU    ($c000<<ESSI_SCI_SHIFT)
                                        ; 1100 0000 0000 0000
                                        ; 110- ---- ---- ---- 4.5x multiplier, max'd out
```

```
                                        ; ---0 0--- ---- ---- no turbo mode, running at max
                                        ; ---- -000 0000 0000 12.288 MHz crystal (default)


SCI_VOL_VALUE    EQU     ($0000<<ESSI_SCI_SHIFT)
                                        ;max volume, DAC does vol control

SCI_MODE_VALUE   EQU     ($0800<<ESSI_SCI_SHIFT) ;allow SDI tests
SCI_MODE_CANCEL_VALUE EQU ($0808<<ESSI_SCI_SHIFT) ;cancel current playback
SCI_AIADDR_VALUE EQU     ($4022<<ESSI_SCI_SHIFT) ;sine sweep test


;MEMORY MAPPED 1053 REGISTERS (accessable via WRAMADDR/WMRAM)
;there a number of 1053-memory-mapped registers that we need to write to:
GPIO_DDR_ADDR    EQU     ($C017<<ESSI_SCI_SHIFT)

I2S_CONFIG_ADDR EQU      ($C040<<ESSI_SCI_SHIFT)
ENDFILLBYTE_ADDR EQU     ($1e06<<ESSI_SCI_SHIFT)



;and we set these registers to enable I2S per the spec:

GPIO_DDR_VALUE     EQU   ($00f0<<ESSI_SCI_SHIFT)   ;set GPIO pins to all output
I2S_CONFIG_VALUE   EQU   ($000d<<ESSI_SCI_SHIFT)   ;set I2S, MCLK, 96khz


; the following deals with data transmission via DMA, etc.

DCR2             EQU     $4a5a50        ; 0100 1010 0101 1010 0101 0000
                                        ; 0--- ---- ---- ---- ---- ---- not enabled (will do later)
                                        ; -1-- ---- ---- ---- ---- ---- interrupt enabled
                                        ; --00 1--- ---- ---- ---- ---- word transfer, DE-enabled
                                        ; ---- -01- ---- ---- ---- ---- normal priority DMA
                                        ; ---- ---0 ---- ---- ---- ---- not continuous
                                        ; ---- ---- 0101 1--- ---- ---- trigger on ESSI0 TX empty
                                        ; ---- ---- ---- -0-- ---- ---- not 3d mode
                                        ; ---- ---- ---- --10 0--- ---- dest (ESSI) no update
                                        ; ---- ---- ---- ---- -101 ---- source (buffer) post-inc
                                        ; ---- ---- ---- ---- ---- 00-- destination in X (I/O)
                                       ; ---- ---- ---- ---- ---- --00 source in X (buffer)


```

```
;constants for our DMA/transfer logic
NEEDS_KICKSTART EQU 1
NO_KICKSTART EQU 0
DREQ_CTL_PRIO EQU %110 ;set to edge-trigger, priority 1, we can transmit up to
                       ; 32 bytes safely due to safety zone, even after DREQ is
                       ; lowered
NO_BUFFER_LEN EQU 0




;the following are helpful constants when dealing with our DAC, the Wolfson
; WM8741. we interface to it via a write-only control interface on ESSI1.

; START DAC SECTION

ESSI1_CRA       EQU     $100806         ; 0001 0000 0000 1000 0000 0110
                                        ; 0--- --0- ---- -000 ---- ---- reserved
                                        ; -0-- ---- ---- ---- ---- ---- SC1 as flag
                                        ; --01 0--- ---- ---- ---- ---- 16b words
                                        ; ---- -0-- ---- ---- ---- ---- align left
                                        ; ---- ---0 0000 ---- ---- ---- on-demand mode
                                        ; ---- ---- ---- 1--- ---- ---- no prescaler
                                        ; ---- ---- ---- ---- 0000 0110 divide by 6+1
                                        ;                               (80/7 = 11.428)

;we use no interrupts with the DAC, because it's fast enough for us just to use
; synchronous changes, with minimal lag.

ESSI1_CRB       EQU     $003c30         ; 0000 0000 0001 1100 0011 0000
                                        ; 0--- ---- ---- ---- ---- ---- no RX ex. interrupt
                                        ; -0-- ---- ---- ---- ---- ---- no TX ex. interrupt
                                        ; --0- ---- ---- ---- ---- ---- no RX slot interrupt
                                        ; ---0 ---- ---- ---- ---- ---- no TX slot interrupt
                                        ; ---- 0--- ---- ---- ---- ---- no RX interrupt
                                        ; ---- -0-- ---- ---- ---- ---- no TX interrupt
                                        ; ---- --0- ---- ---- ---- ---- RX disabled
                                        ; ---- ---0 ---- ---- ---- ---- TX0 (STD) disabled (enabled later)
                                        ; ---- ---- 0--- ---- ---- ---- TX1 (SC0) disabled
                                        ; ---- ---- -0-- ---- ---- ---- TX2 (SC1) disabled
```

```
                                        ; ---- ---- --1- ---- ---- ---- network (on-demand) mode
                                        ; ---- ---- ---1 ---- ---- ---- synchronous mode
                                        ; ---- ---- ---- 1--- ---- ---- clock out data on falling edge
                                        ; ---- ---- ---- -1-- ---- ---- negative edge triggers word (positive is end of word)
                                        ; ---- ---- ---- --0- ---- ---- frame sync w/ first bit
                                        ; ---- ---- ---- ---0 0--- ---- word-long sync to get positive edge at end
                                        ; ---- ---- ---- ---- -0-- ---- MSB shifted first
                                        ; ---- ---- ---- ---- --1- ---- internal clock
                                        ; ---- ---- ---- ---- ---1 ---- SC2 is an output
                                        ; ---- ---- ---- ---- ---- 0--- SC1 is GPIO, don't care
                                        ; ---- ---- ---- ---- ---- -0-- SC0 is GPIO, don't care
                                        ; ---- ---- ---- ---- ---- --00 don't care, don't use flags

ESSI1_PCRD      EQU     $00002c         ; 0000 0000 0000 0000 0010 1100
                                        ; 0000 0000 0000 0000 00-- ---- reserved
                                        ; ---- ---- ---- ---- --1- ---- TX enabled
                                        ; ---- ---- ---- ---- ---0 ---- RX is GPIO
                                        ; ---- ---- ---- ---- ---- 1--- CLK enabled
                                        ; ---- ---- ---- ---- ---- -1-- SC2 enabled
                                        ; ---- ---- ---- ---- ---- --0- SC1 is GPIO
                                        ; ---- ---- ---- ---- ---- ---0 SC0 is GPIO

;see errata for mask 4H80G, we find that GPIO pins need to be programmed to output
; anyway. this is probably why our shit's not working OTL.

ESSI1_PRRD      EQU     $00003f         ; 0000 0000 0000 0000 0011 1111
                                        ; 0000 0000 0000 0000 00-- ---- reserved
                                        ; ---- ---- ---- ---- --11 1111 all pins output


; to write a command to the WM8741 is as simple as moving in the appropriate command
; to ESSI1 TX0 and sleeping appropriately to ensure transmission.

;these shifts are based on the fact that ESSI takes left-aligned data
```

```
DAC_ADDR_SHIFT   EQU     9+8      ;we shift 9 bits over as our addresses comprise
                                  ; the first 7 bits of our data word
DAC_DATA_SHIFT   EQU     8        ;the data goes in the last 8 bits of the 16 bit word

;here are a bunch of appropriately shifted and masked addresses to use with our system
DAC_LLSB_ADDR    EQU     ($0<<DAC_ADDR_SHIFT)
DAC_LMSB_ADDR    EQU     ($1<<DAC_ADDR_SHIFT)
DAC_RLSB_ADDR    EQU     ($2<<DAC_ADDR_SHIFT)
DAC_RMSB_ADDR    EQU     ($3<<DAC_ADDR_SHIFT)
DAC_VOL_ADDR     EQU     ($4<<DAC_ADDR_SHIFT)
DAC_FORMAT_ADDR  EQU     ($5<<DAC_ADDR_SHIFT)
DAC_FILTER_ADDR  EQU     ($6<<DAC_ADDR_SHIFT)
DAC_MODE1_ADDR   EQU     ($7<<DAC_ADDR_SHIFT)
DAC_MODE2_ADDR   EQU     ($8<<DAC_ADDR_SHIFT)
DAC_RESET_ADDR   EQU     ($9<<DAC_ADDR_SHIFT)
DAC_ADDCTL_ADDR   EQU    ($20<<DAC_ADDR_SHIFT)

;here are a bunch of values we expect to initialize with
DAC_VOL_CONTROL  EQU     ($07<<DAC_DATA_SHIFT)
                                          ; 0001 0111
                                          ; 0--- ---- not muted
                                          ; -00- ---- ZFLAG from both channels
                                          ; ---0 ---- no analog mute on ZFLAG
                                          ; ---- 0--- no mute
                                          ; ---- -1-- control both channels w/ left atten
                                          ; ---- --1- anti-clipping enable
                                          ; ---- ---1 ramp vol changes
DAC_FILTER_CONTROL EQU   ($01<<DAC_DATA_SHIFT)
                                          ; 0000 0011
                                          ; 0--- ---- normal ZFLAG
                                          ; -00- ---- de-emph off
                                          ; ---0 0--- DSD filter (don't care)
                                          ; ---- -001 filter #2: minimum-phase linear

DAC_MODE_CONTROL_1 EQU   ($20<<DAC_DATA_SHIFT)
                                          ; 0010 0000

                                          ; 0--- ---- no 8fs mode
                                          ; -01- ---- medium sample rate (96KHz) - 192 can be unstable
                                          ; ---0 00-- autodetect MCLK
                                          ; ---- --00 PCM mode

;volume control stuff
ATTEN_MASK         EQU   $3ff             ;WM8741 has space for 10 bits of volume
ATTEN_REG_MASK     EQU   $1f              ;5 bits in the LSB of two registers
ATTEN_LATCH_MASK   EQU   $20              ;this bit latches all volume changes
ATTEN_STEP         EQU   16               ;step in 2dB increments
MIN_ATTEN          EQU   0                ;no attenuation == minimum
MAX_ATTEN          EQU   $3ff             ;max attenuation = -127.5/mute
INIT_ATTEN         EQU   $dc              ; ~-25 db to start, reasonable volume

```

```
        opt     noidw
        opt so,xr
        page 132,66,3,3

;
; sound.asm
;
; This file offers interfacing code to the sound subsystem: the MP3 decoder and
; the DAC.
;

        section sound
 INCLUDE 'DSP_EQU.inc'
 INCLUDE 'sound.inc'
 INCLUDE 'macros.inc'
 INCLUDE 'general.inc'
        org     x:

;VOLATILE declarations go here

; sound_atten is our volume attenuation in 10 bits, in -0.125dB steps.
sound_atten:
        ds      1


;here are all our pointers/etc for the audio update system
curbuf:
        ds      1               ;a pointer to the current buffer we're playing
curbuflen:
        ds      1               ;length of the current buffer

nextbuf:
        ds      1               ;a pointer to the next buffer (double buffering)
nextbuflen:
        ds      1               ;length of the next buffer

usedbuf:
        ds      1               ;a pointer to a used buffer

mp3_kickstart:
        ds      1               ;this is set non-zero if the mp3 decoder can
                                ; accept data, and was idle previously

;CONSTANT declarations go here

        org     p:              ;we want to be located in program space

;PROGRAM (function) declarations go here

; Fsetup_mp3
; Functional Specification
;
; Description:      This function inits the MP3 decoder chip, synchronously.
; It should be run when no interrupts are enabled.
;
```

```
; Operation:        This function sets up ESSI0 to interface with the MP3
; decoder chip (VLSI solutions VS1053b) in New Mode, and sets the clock
; registers as appropriate. It also enables I2S output so we can get
; our DAC outputting something.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fsetup_mp3
Fsetup_mp3:                         ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl
        PUSH_ACC a


        ;initialize the ESSI 0 control registers
        move    #>ESSI0_CRA_1ST,r0
        move    r0,x:M_CRA0
        move    #>ESSI0_CRB_1ST,r0
        move    r0,x:M_CRB0

        ;now configure the ESSI 0 pins as active and not GPIO
```

```
        move    #>ESSI0_PCRC_SCI,r0
        move    r0,x:M_PCRC
        move    #>ESSI0_PRRC,r0
        move    r0,x:M_PRRC
        move    #>ESSI0_PDRC,r0
        move    r0,x:M_PDRC

        ;we load first data
        move    #>$ffff00,r0
        move    r0,x:M_TX00
        ; (TX02 control XCS)
        move    #XCS_HIGH,r0
        move    r0,x:M_TX02

        ;now enable transmitters
        bset    #M_SSTE0,x:M_CRB0
        bset    #M_SSTE2,x:M_CRB0
        bset    #M_SSRE,x:M_CRB0

        ;we wait for mp3 to become ready again
        move    #>SET_CLOCK_WAIT,r0
        jsr     sleep

        ;we now need to do several things:
        ; 1) set the clock multiplier on the MP3 to to 3 or so
        ;    (12MHz clock is really slow)
        ; 2) set the I2S output to enable
        ; 3) set volume to max (DAC handles volume)
        ; 4) in the future we'll add patches (like FLAC decoder)

        ; 1) we set the clock multiplier
        move    #>(SCI_WRITE_CMD|SCI_CLOCKF),r0
        move    #>SCI_CLOCKF_VALUE,r1
        jsr     write_sync_mp3_sci

        ;we wait for mp3 to become ready again
        move    #>SET_CLOCK_WAIT,r0
        jsr     sleep

        ;we can now use faster clock speeds, so set it


        nop
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*   ;wait until transmitted, we disable then
                                      ;re-enable

        move    #>ESSI0_CRA_SCI,r0   ;set new values of CRB/CRA
        move    r0,x:M_CRA0
        move    #>ESSI0_CRB_SCI,r0
        move    r0,x:M_CRB0

        ;as before, we now load shill data:
        move    #>$ffff00,r0
        move    r0,x:M_TX00
        move    #XCS_HIGH,r0
```

```
        move    r0,x:M_TX02

        ;now enable transmitters
        bset    #M_SSTE0,x:M_CRB0
        bset    #M_SSTE2,x:M_CRB0
        bset    #M_SSRE,x:M_CRB0

        ; 2) enable I2S output

        ;to enable I2S, we turn GPIO to output and turn on I2S
        ;set GPIO flags:
        move    #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0       ;we need to set where
                                                        ; to write to in 1053
        move    #>GPIO_DDR_ADDR,r1
        jsr     write_sync_mp3_sci
        move    #>SET_WRAMADDR_WAIT,r0
        jsr     sleep

        move    #>(SCI_WRITE_CMD|SCI_WRAM),r0           ;now write content
        move    #>GPIO_DDR_VALUE,r1
        jsr     write_sync_mp3_sci
        move    #>SET_WRAM_WAIT,r0
        jsr     sleep

        ;set I2S enable:
        move    #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0       ;we need to set where
                                                        ; to write to in 1053
        move    #>I2S_CONFIG_ADDR,r1
        jsr     write_sync_mp3_sci
        move    #>SET_WRAMADDR_WAIT,r0
        jsr     sleep

        move    #>(SCI_WRITE_CMD|SCI_WRAM),r0           ;now write content
        move    #>I2S_CONFIG_VALUE,r1
        jsr     write_sync_mp3_sci
        move    #>SET_WRAM_WAIT,r0
        jsr     sleep


        ; 3) set analog out to max volume
        move    #>(SCI_WRITE_CMD|SCI_VOL),r0
        move    #>SCI_VOL_VALUE,r1
        jsr     write_sync_mp3_sci
        move    #>SET_VOL_WAIT,r0
        jsr     sleep


        ; 4) instead of patches, we set SDI test mode for sine waves.
        ;we first set SM_TESTS = 1
        move    #>(SCI_WRITE_CMD|SCI_MODE),r0
        move    #SCI_MODE_VALUE,r1
        jsr     write_sync_mp3_sci
        move    #>SET_MODE_WAIT,r0
        jsr     sleep

        ;then write AIADDR to enable the sine wave test
;        move    #>(SCI_WRITE_CMD|SCI_AIADDR),r0
```

```
;        move    #SCI_AIADDR_VALUE,r1
;        jsr     write_sync_mp3_sci
;        move    #>SET_AIADDR_WAIT,r0
;        jsr     sleep

         ; 5) we now switch to data mode
         jsr     mp3_to_data_mode

        ;and we setup the DMA control registers as we'll be using
        ; them in the future
        move    #>DCR2,r0
        move    r0,x:M_DCR2
        move    #>M_TX00,r0
        move    r0,x:M_DDR2

        ;we also install the interrupt handlers we'll be using

        move    #>$0bf080,r0    ;unconditional branch
        move    r0,p:I_DMA2
        move    #end_mp3_dma_handler,r1 ;install DMA handler
        move    r1,p:(I_DMA2+1)

        ;install the pair of DREQ handlers
        move    #dreq_asserted_handler,r1
        move    r0,p:I_IRQB
        move    r1,p:(I_IRQB+1)

        move    #dreq_deasserted_handler,r1
        move    r0,p:I_IRQC
        move    r1,p:(I_IRQC+1)

        ;we now clear XCS
        move #0,r1

;this commented out portion implements a SDI test vector (5133Hz sine)

;        nop     ;wait for delay
;        nop
;        nop

;        move    #>$53ef,r1
;        move    r1,x:M_TX00          ;shill write to TX00
;        nop     ;wait for delay
;        nop
;        nop
;        jclr    #M_TDE,x:M_SSISR0,*   ;wait until transmitted

;        move    #>$6e7e,r1
;        move    r1,x:M_TX00          ;shill write to TX00
;        nop     ;wait for delay
;        nop
;        nop
;        jclr    #M_TDE,x:M_SSISR0,*   ;wait until transmitted

;        move    #>$0,r1
;        move    r1,x:M_TX00          ;shill write to TX00
;        nop     ;wait for delay
```

```
;         nop
;         nop
;         jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted

;         move    #>$0,r1
;         move    r1,x:M_TX00            ;shill write to TX00
;         nop     ;wait for delay
;         nop
;         nop
;         jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted





;end of body code, now restore all registers
        POP_ACC a
        move    ssl,r1
        move    ssh,r0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts     ;return from subroutine

;end Fsetup_mp3


; mp3_to_data_mode
; Functional Specification
;
; Description:      mp3_to_data_mode changes ESSI0 to enter data
; mode, so that we can transmit MP3 data to it.
;
; Operation:        mp3_to_data_mode changes the mode to enable 16-bit
; data words and reconfigures PC2 to function as a frame sync, thus giving
; us an easy time. It also disables the SC1 (XCS) transmitter and sets it
; as a GPIO pin, making it easier for us.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
```

```
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    This function turns off ESSI0's interrupts. It does not
; enable the DREQ/IRQ1 interrupt.
;
; Revision History: 2011/03/28    Raymond Jimenez    first revision
;
        global  mp3_to_data_mode
mp3_to_data_mode:                       ;assembly entry point

        ;save all registers we use so we can restore later
        move    r0,ssh

        ;we make sure all transmits are done:
        nop
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*   ;wait until transmitted

                ;need to temporarily disable ESSI pins otherwise they retain functionality
        move    #>0,r0
        move    r0,x:M_PCRC

                ;we then reconfigure ESSI0 to give us automatic frame syncs
        ; (auto XDCS)
        move    #ESSI0_CRA_DATA,r0
        move    r0,x:M_CRA0
        move    #ESSI0_CRB_DATA,r0
        move    r0,x:M_CRB0

        ;then, we set the GPIO pins as necessary
        move    #>ESSI0_PCRC_DATA,r0
        move    r0,x:M_PCRC

        ;load shill data
        move    #>0,r0
        move    r0,x:M_TX00

        ;then enable transmitters
        bset    #M_SSTE0,x:M_CRB0

        ;and that's it

        ;end of body code, now restore all registers
        move    ssh,r0

        nop     ;pipeline cache issue (see errata)
```

```
        rts     ;return from subroutine
;end mp3_to_data_mode

; mp3_to_control_mode
; Functional Specification
;
; Description:      mp3_to_control_mode changes ESSI0 to control mode,
; so we can use write_sync_mp3_sci/read_sync_mp3_sci.
;
; Operation:        mp3_to_control_mode changes the ESSI0/GPIO registers
; such that we can use it to transmit control words now.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    This function turns off ESSI0's interrupts. It does not
; enable/disable the DREQ/IRQ1 interrupt.
;
; Revision History: 2011/03/28    Raymond Jimenez    first revision
;
        global  mp3_to_control_mode
mp3_to_control_mode:                        ;assembly entry point

        ;save all registers we use so we can restore later
        move    r0,ssh

        ;we manually disable DMA.
        bclr    #M_DE,x:M_DCR2
        ;and we wait for the DMA transfers to finish
        jset    #M_DACT,x:M_DSTR,*

        ;we make sure all transmits are done:
        nop
        nop
```

```
        nop
        jclr    #M_TDE,x:M_SSISR0,*   ;wait until transmitted

                ;need to temporarily disable ESSI pins otherwise they retain functionality
        move    #>0,r0
        move    r0,x:M_PCRC


        ;we then reconfigure ESSI0 to give us automatic frame syncs
        ; (auto XDCS)
        move    #ESSI0_CRA_SCI,r0
        move    r0,x:M_CRA0
        move    #ESSI0_CRB_SCI,r0
        move    r0,x:M_CRB0

        ;first, we set the GPIO pins as necessary
        move    #>ESSI0_PCRC_SCI,r0
        move    r0,x:M_PCRC

        ;we load first data
        move    #>$ffff00,r0
        move    r0,x:M_TX00
        ; (TX02 control XCS)
        move    #XCS_HIGH,r0
        move    r0,x:M_TX02

        bset    #M_SSTE0,x:M_CRB0
        bset    #M_SSTE2,x:M_CRB0
        bset    #M_SSRE,x:M_CRB0


        ;and that's it

        ;end of body code, now restore all registers
        move    ssh,r0

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine
;end mp3_to_control_mode


; write_sync_mp3_sci
; Functional Specification
;
; Description:      write_sync_mp3_sci transmits an SCI command to the mp3
; decoder, synchronously. command is guaranteed to be transmitted but the
; mp3 decoder is not guaranteed to be ready after this. (you must insert
; a sleep after to account for worst-case wait, or wait for DREQ to become high)
;
; ESSI0 interrupts must be disabled before invocation.
;
; Operation:        write_sync_mp3_sci first changes the ESSI0 TX2 to indicate
; XCS is selected. it then checks to make sure the TX buffers are empty, and
; transmits the desired control words as necessary.
;
```

```
; Arguments:        r0 - SCI command and address to write, left aligned.
;                        r0[23:16] - should be $02 for write
;                        r0[15:8]  - SCI address
;                        r0[7:0]   - reserved, write 0 for compat
;
;                   r1 - SCI data to write
;                        r1[23:8]  - SCI data to write
;                        r1[7:0]   - reserved, write 0 for compat
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    We require the strange bitfield structure of r0/1 in order to
; prevent us from having to transfer into a, and then rotate. it's much faster
; this way, and since we mostly transmit constants to SCI, we can just
; pre-compose commands via macros.
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  write_sync_mp3_sci
write_sync_mp3_sci:                         ;assembly entry point

        ;save all registers we use so we can restore later
        PUSH_ACC a
        move    r2,ssh

        ;note that these sections are time-critical, so we block on them.
        ; (this is another reason to up the clock speed on our serial transmits)
        START_CRITICAL

        ;we first make sure the transmitter is empty
        jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
        move    r0,x:M_TX00            ;transmit first 16 bits: command, address

```

```
        ;we first want to make sure XCS is turned on for these words,
        ; so set up TX10/TX20 appropriately
        move    #XCS_HIGH,r2
        move    r2,x:M_TX02


        ;we first make sure the transmitter is empty

        ;we have a two-cycle pipeline delay after we write these registers
        ; see DSP3609 user manual, pg 7-31
        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
        move    r0,x:M_TX00            ;transmit first 16 bits: command, address

        ;we first want to make sure XCS is turned on for these words,
        ; so set up TX10/TX20 appropriately
        move    #XCS_LOW,r2
        move    r2,x:M_TX02


        ;and then we move in our words:

        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted

        move    r1,x:M_TX00            ;next 16 bits, also filling TX10/TX20
        move    #XCS_LOW,r2
        move    r2,x:M_TX02



        ;we now clear XCS

        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted

        move    r1,x:M_TX00            ;shill write to TX00

        move    #XCS_HIGH,r2
        move    r2,x:M_TX02

        END_CRITICAL

        ;end of body code, now restore all registers
        move    ssh,r2
        POP_ACC a

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine
;end write_sync_mp3_sci

```

```

; read_sync_mp3_sci
; Functional Specification
;
; Description:      read_sync_mp3_sci reads from the mp3 decoder, selecting a
; register/address to read from.
;
; ESSI0 interrupts must be disabled before invocation.
;
; Operation:        write_sync_mp3_sci first
;
; Arguments:        r0 - SCI command and address to write, left aligned.
;                        r0[23:16] - should be $03 for read
;                        r0[15:8]  - SCI address
;                        r0[7:0]   - reserved, write 0 for compat
;
; Return Values:    a1 - data word transmitted by mp3 decoder
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: a
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    We require the strange bitfield structure of r0/1 in order to
; prevent us from having to transfer into a, and then rotate. it's much faster
; this way, and since we mostly transmit constants to SCI, we can just
; pre-compose commands via macros.
;
; Revision History: 2011/03/27    Raymond Jimenez    first revision
;
        global  read_sync_mp3_sci
read_sync_mp3_sci:                      ;assembly entry point

        ;save all registers we use so we can restore later
        move    r2,ssh


        ;note that these sections are time-critical, so we block on them.
        ; (this is another reason to up the clock speed on our serial
```

```
        transmits)
        START_CRITICAL

        ;we first make sure the transmitter is empty
        jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
        move    r0,x:M_TX00            ;transmit first 16 bits: command, address

        ;we first want to make sure XCS is turned on for these words,
        ; so set up TX10/TX20 appropriately
        move    #XCS_HIGH,r2
        move    r2,x:M_TX02


        ;we first make sure the transmitter is empty

        ;we have a two-cycle pipeline delay after we write these registers
        ; see DSP3609 user manual, pg 7-31
        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
        move    r0,x:M_TX00            ;transmit first 16 bits: command, address

        ;we first want to make sure XCS is turned on for these words,
        ; so set up TX10/TX20 appropriately
        move    #XCS_LOW,r2
        move    r2,x:M_TX02


        ;and then we move in our words:

        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted

        move    r0,x:M_TX00            ;shill write to get data; don't care about
                                       ; what data we transmit.
        move    #XCS_LOW,r2
        move    r2,x:M_TX02


        nop     ;wait for delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set


        move    x:M_RX0,a              ;read recieved data - we expect it to be
                                       ; read as it's completed so it doesn't
                                       ; get overwritten. (right after TDE
                                       ; is guaranteed to be one serial word)



        ;we now clear XCS
```

```
        jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted

        move    r1,x:M_TX00            ;shill write to TX00

        move    #XCS_HIGH,r2
        move    r2,x:M_TX02

        END_CRITICAL

        ;end of body code, now restore all registers
        move    ssh,r2

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine
;end read_sync_mp3_sci


; Faudio_play
; Functional Specification
;
; Description:      Faudio_play sets up the interrupts and appropriate variables
; for the audio playback system.
;
; Operation:        Faudio_play first calls Faudio_halt, which synchronously
; stops audio playback. We then set up the registers again, and then start
; the file playing by setting up the interrupts and letting the ESSI0 TX empty
; trigger run the DMA request.
;
; Arguments:        a1 - pointer to new buffer to use
;                   b1 - length of buffer at pointer
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
```

```
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Faudio_play
Faudio_play:                           ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl
        PUSH_ACC b

        ;we first halt any playback that might be happening
;        jsr     Faudio_halt

        ;we are now guaranteed that the VLSI can take our data, DREQ must be high

        ;now, we need to setup the DMA control registers and our buffers
        move    #>NO_BUFFER_LEN,r0
        move    r0,x:nextbuf    ;we clear out these variables to prevent confusion
        move    r0,x:nextbuflen
        move    r0,x:usedbuf

        move    a1,x:curbuf     ;and move in our actual buffer values
        move    b1,x:curbuflen

        ;process the mp3 buffer
        move    a1,r0   ;prepare arguments for prepare_mp3_buffer
        move    b1,r1
        jsr     process_mp3_buffer

        move    #>NO_KICKSTART,r0
        move    r0,x:mp3_kickstart ;we're starting the writing process, so no
                                   ;kickstart

        move    x:curbuf,a    ;and move in our actual buffer values
        move    x:curbuflen,b

        move    a1,x:M_DSR2     ;we now setup the DMA pointers
        ;unfortunately, DMA transfers counter + 1 in total, so we subtract:
        sub     #>1,b
        move    b1,x:M_DCO2     ;tell the DMA how many words from this buffer


        ;before DMA can go, we need to set enable the interrupts on DREQ off/on
        move    x:M_IPRC,b      ;we need to OR in our new priority bits
        and     #>((~M_IBL)&(~M_ICL)&(~M_D2L)),b ;clear out bits there
        or      #>(DREQ_CTL_PRIO<<M_IBL0)|(DREQ_CTL_PRIO<<M_ICL0)|(3<<M_D2L0),b

                                ;set our new bits
```

```
        move    b1,x:M_IPRC

        ;and then finally, we enable the DMA and get going

        bset    #M_DE,x:M_DCR2  ;set enable, since DMA is DE-enabled

        ;at this point we should be going at full tilt:
        ; if the buffer runs out, the DMA done interrupt will handle it
        ; if DREQ goes low, the DREQ-done handler will disable DMA
        ; if DREQ goes high after low, the DREQ-more handler will enable DMA

        ;end of body code, now restore all registers
        POP_ACC b
        move    ssl,r1
        move    ssh,r0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Faudio_play

; Faudio_halt
; Functional Specification
;
; Description:      Faudio_halt turns off the DMA and other interrupts
; so that we can tell the decoder to halt, synchronously. This function does
; not return until audio playback has stopped.
;
; Operation:        Faudio_halt follows the procedure laid out in the 1053
; manual: it first reads endFillByte, sends 2052 bytes of endFillByte,
; then sets SM_CANCEL, sends another 2052 bytes of endFillByte, and then
; returns.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
```

```
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Faudio_halt
Faudio_halt:                       ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        PUSH_ACC a
        PUSH_ACC b
        move    r0,ssh
        move    r1,ssl
        move    x0,ssh


        ;disable interrupts, all reads from here on are synchronous
        move    x:M_IPRC,a      ;move to a so we can clear out bits
        and     #>((~M_IBL)&(~M_ICL)&(~M_D2L)),a ;clear out bits there (IRQB/IRQC are DRQ)
        move    a1,x:M_IPRC

        ;we switch to control mode to read endFillByte
        jsr     mp3_to_control_mode


        move    #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0           ;set address
        move    #>ENDFILLBYTE_ADDR,r1
        jsr     write_sync_mp3_sci
        move    #>SET_WRAMADDR_WAIT,r0
        jsr     sleep

        move    #>(SCI_READ_CMD|SCI_WRAM),r0           ;read data
        jsr     read_sync_mp3_sci
        ;we now have endFillByte in a1[15-8]. let's duplicate it
        ; to [7-0] for ease of use.
        and     #>$00ff00,a            ;make sure only one byte is set
        move    a1,b                   ;we use b as a scratch register
        lsl     #8,b                   ;put a copy in the top byte
        move    b1,x0
        or      x0,a                    ;now a has a copy of two bytes of endfillbyte

        jsr     mp3_to_data_mode       ;now we need to write 2052 bytes
```

```

        do      #2052,write_endfillbyte
        move    a1,x:M_TX00             ;write endfillbyte
        nop     ;account for pipeline delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*   ;keep on jumping here until transmitted

        nop     ;spacing nops
        nop
        nop
write_endfillbyte:

        jsr     mp3_to_control_mode   ;we now need to shift to control mode to
                                      ; set SM_CANCEL
        move    #>(SCI_WRITE_CMD|SCI_MODE),r0
        move    #>SCI_MODE_CANCEL_VALUE,r1
        jsr     write_sync_mp3_sci
        move    #>SET_MODE_WAIT,r0
        jsr     sleep
        ;now that we've written cancel, we transmit 2052 bytes of endfillbyte again

        jsr     mp3_to_data_mode

        do      #2052,write_endfillbyte2
        move    a1,x:M_TX00             ;write endfillbyte
        nop     ;account for pipeline delay
        nop
        nop
        jclr    #M_TDE,x:M_SSISR0,*   ;keep on jumping here until transmitted

        nop     ;spacing nops
        nop
        nop
write_endfillbyte2:

        ;we're now done.

;end of body code, now restore all registers
        move    ssh,x0
        move    ssl,r1
        move    ssh,r0
        POP_ACC b
        POP_ACC a


        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Faudio_halt
```

```
; end_mp3_dma_handler
; Functional Specification
;
; Description:      end_mp3_dma_handler takes care of switching to the next
; buffer at the end of a DMA transfer to the mp3 decoder.
;
; Operation:        end_mp3_dma_handler handles the case when we get to the end
; of a DMA transfer, and so we need more data. We first check to see if we have
; a valid next buffer; if so, we swap it out and go for it.
;
; if there is no valid next data buffer, we set mp3_kickstart to NEEDS_KICKSTART
; and we indicate that our buffer has been used up by moving it into usedbuf.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  end_mp3_dma_handler
end_mp3_dma_handler:                        ;assembly entry point

        ;save all registers we use so we can restore later
        PUSH_ACC a
        move x0,ssh
        move r0,ssl

        ;we first clear out interrupt enable bit to prevent
        ; multiple interrupts
        bclr    #M_DIE,x:M_DCR2             ;clear out interrupt bit
```

```

        ;since we're called if DE has also been hand-deasserted
        ; (as in DREQ stopping our data transfer), we
        ; check to see if we were at the end of the transfer by comparing
        ; the buffer length to DCR+1.
        move    x:M_DCO2,a
        add     #>1,a               ;number of transfer done is DCR+1, so we
                                    ; compensated for this
        move    x:curbuflen,x0
        cmp     x0,a

        jeq     end_mp3_true_end    ;if we got here by an artificial DE
                                    ;disable, we skip our next-buffer logic
end_mp3_false_end:
        ;we just set the DIE bit and go on our way, if false

        ;re-enable DMA channel interrupts, in all cases
        bset    #M_DIE,x:M_DCR2
        jmp     end_mp3_dma_handler_done

end_mp3_true_end:
        ;we're now sure that we're operating from the end of the buffer.

        move    x:curbuf,r0                 ;we indicate this buffer's been used
        nop
        move    r0,x:usedbuf

        ;let's check if there's more data in the next buffer.
        move    x:nextbuflen,a
        cmp     #>NO_BUFFER_LEN,a           ;check if we have no buffer
        jeq     end_mp3_dma_no_buf

end_mp3_dma_buf: ;we're now guaranteed to have a buffer to move in
        move    x:nextbuf,r0
        nop
        move    r0,x:M_DSR2                 ;load the new source buffer
        move    r0,x:curbuf

        move    x:nextbuflen,a              ;we now subtract one to get DCO
        move    a1,x:curbuflen
        sub     #>1,a
        move    a1,x:M_DCO2                 ;load in the new counter

        move    #>NO_BUFFER_LEN,r0          ;and we mark that we have no next
        move    r0,x:nextbuf
        move    r0,x:nextbuflen

        ;also, we mark that we don't need a kickstart, we've done it
        move    #>NO_KICKSTART,r0
        move    r0,x:mp3_kickstart

        ;we now enable the DMA channel again
        bset    #M_DIE,x:M_DCR2
        bset    #M_DE,x:M_DCR2

        jmp     end_mp3_dma_handler_done

```

```
end_mp3_dma_no_buf: ;we're guaranteed to have no buffers to move in
        ;since we have no next buffer, we mark the current buffer
        ; used as well.
        move    x:curbuf,r0
        nop
        move    r0,x:usedbuf

        ;we invalidate the current buffer by setting it to NO_BUFFER_LEN
        move    a1,x:curbuflen             ;contains NO_BUFFER_LEN

        ;we will need a kickstart after this, so we set it
        move    #>NEEDS_KICKSTART,r0
        move    r0,x:mp3_kickstart

        ;that should be it!
        ;re-enable DMA channel interrupts, in all cases
        bset    #M_DIE,x:M_DCR2


end_mp3_dma_handler_done:

        ;end of body code, now restore all registers
        move    ssl,r0
        move    ssh,x0
        POP_ACC a

        nop     ;pipeline cache issue (see errata)
        rti     ;return from interrupt
;end end_mp3_dma_handler

; dreq_asserted_handler
; Functional Specification
;
; Description:      dreq_asserted_handler takes care of the situation when
; the 1053's DREQ has been freshly asserted.
;
; Operation:        dreq_asserted_handler sets the DMA enable bit on our
; DMA channel. we only see a negative-going edge on DREQ if we are newly
; renewing an old data transfer, because DREQ should start high coming out
; of control mode.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
```

```
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  dreq_asserted_handler
dreq_asserted_handler:                          ;assembly entry point

        ;save all registers we use so we can restore later
        PUSH_ACC a

        ;we check if there's good data to be transmitted
        move    x:curbuflen,a
        cmp     #>NO_BUFFER_LEN,a
        jeq     dreq_asserted_no_buffer

dreq_asserted_buffer:
        bset    #M_DE,x:M_DCR2    ;set our DMA channel to go
        jmp     dreq_asserted_done

dreq_asserted_no_buffer:
        move    #>NEEDS_KICKSTART,a
        move    a1,x:mp3_kickstart

dreq_asserted_done:
        ;end of body code, now restore all registers
        POP_ACC a

        nop     ;pipeline cache issue (see errata)
        rti     ;return from interrupt

;end dreq_asserted_handler

; dreq_deasserted_handler
; Functional Specification
;
; Description:      dreq_deasserted_handler takes care of the case when the
1053
; can't receive any more data.
;
; Operation:        It disables the DMA channel by clearing DE.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
```

```
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  dreq_deasserted_handler
dreq_deasserted_handler:                        ;assembly entry point

        ;save all registers we use so we can restore later

        bclr    #M_DE,x:M_DCR2                  ;disable the current DMA transfer.
        nop
        nop
        nop  ;pipeline delays
        ;wait for it to finish its current word transfer
;        jset    #M_DACT,x:M_DSTR,*

        ;end of body code, now restore all registers

        nop     ;pipeline cache issue (see errata)
        rti     ;return from interrupt
;end dreq_deasserted_handler


; Fupdate
; Functional Specification
;
; Description:      Fupdate is to be called periodically, and gives new data
; to the DMA if necessary. it returns true (non-zero) if more data was needed
; and the passed buffer is a used buffer. it returns false (zero) if no more
; data can be taken.
;
; Operation:
;
; we check: is nextbuf occupied?
; yes:
; case #1: curbuf is occupied, nextbuf occupied, anybuf passed
```

```
;  return false -- no space
;
; no: we then check, is curbuf occupied?
;
; no:
; case #2: curbuf is empty, nextbuf is empty, usedbuf (formery curbuf) is passed
;  -- this is the kickstart scenario, after an underrun
;  we return true, since we need more data
;
; case #3: curbuf is empty, nextbuf is empty, not-usedbuf is passed
;  -- happens after an underrun (2nd step) or something like that, where we
;     missed the polling loop
;  we return false, after placing the new data in and starting DMA
;
; yes:
; case #4: curbuf is occupied, nextbuf empty, curbuf passed
;  return true -- we've used data and have space for more
;  but, don't do anything
;
; case #5: curbuf is occupied, nextbuf is empty, somebuf is passed
;  return false -- we are accepting new data
;  we place somebuf into curbuf, and let DMA continue
;
;
; Arguments:        a1 - buffer passed, may be new data or old
;                   b1 - length of this passed buffer
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fupdate
```

```
Fupdate:                        ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        PUSH_ACC b
        move    x0,ssh
        move    x1,ssl

        move    r0,ssh
        move    r1,ssl

        ;we can only cmp to x/y and accumulators, so let's
        ;move the arguments given into x0 and x1:
        move    a1,x0           ;move buffer address into x0
        move    a1,r0
        move    b1,x1           ;move buffer length into x1
        move    b1,r1

 START_CRITICAL
        ;we check if both curbuf and nextbuf are occupied
        move    x:nextbuflen,b
        cmp     #>NO_BUFFER_LEN,b
        jeq     update_empty_nextbuf   ;if nextbuf is empty, we have more options

update_full_buffers:
        ;scenario #1: full nextbuf, curbuf
        ;but, if the buffer doesn't fit, we must acquit
        ; (we have no space to handle new data)
        move    #>FALSE,a              ;set our return value
        jmp     update_end

update_empty_nextbuf:
        ;there are four possible scenarios here. we check occupancy of curbuf
        ; to decide our route
        move    x:curbuflen,b
        cmp     #>NO_BUFFER_LEN,b
        jne     update_occupied_curbuf ;we have something in curbuf

        ;scenario #2/3: empty curbuf
        ; this is a TX underrun, so we get these two scenarios out of the way

        ;to distinguish between #2 & #3, we need to check passed buf with usedbuf
update_empty_curbuf:
        move    x:usedbuf,b
        cmp     x0,b            ;check if we've used this buffer already
        jne     update_empty_curbuf_new

;update_empty_curbuf_used: ;scenario #2: we have empty buffers, and this
;buffer's old
        move    #>TRUE,a
```

```
        jmp     update_end

update_empty_curbuf_new: ;scenario #3: we have all empty buffers, and new data
        move    x0,x:curbuf    ;place in the new buffer
        move    x1,x:curbuflen
        jsr     process_mp3_buffer ;and process the buffer when we place it in
                                   ;(r0-r1 set up top)

        move    x0,x:M_DSR2    ;also initialize DMA source/count
        move    x1,b           ;we need to account for DMA transfers = DCO + 1
        sub     #>1,b
        move    b1,x:M_DCO2

        ;we now need to activate DMA to act on this new data
        bset    #M_DE,x:M_DCR2 ;activate

        move    #>FALSE,a
        jmp     update_end

        ;scenario #4/5: we have current data, but could accept more
        ; to differentiate the two, we need to compare passed to curbuf
update_occupied_curbuf:
        move    x:curbuf,b
        cmp     x0,b
        jne     update_occupied_curbuf_new

update_occupied_curbuf_old: ;scenario #4: we have data in curbuf, and passed old data
        move    #>TRUE,a    ;we need more new, not old data
        jmp     update_end

update_occupied_curbuf_new: ;scenario #5: we have new data in passed buf.
        ;process our data first:
        jsr     process_mp3_buffer
        move    x0,x:nextbuf    ;we change the next pointer to reflect new data
        move    x1,x:nextbuflen
        ;DMA magic handles the rest

        move    #>FALSE,a
        jmp     update_end



update_end:

 END_CRITICAL

;end of body code, now restore all registers
        move    ssl,r1
        move    ssh,r0


        move    ssl,x1
        move    ssh,x0
        POP_ACC b


```

```
        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Fupdate

; process_mp3_buffer
; Functional Specification
;
; Description:      process_mp3_buffer processes each byte of the
; mp3 buffer given to it. it assumes that the data is in the lower
; 16 bits of the words given to it, and that we send to ESSI via
; DMA.
;
; Operation:        we copy the word into a1 and b1, and left-align the
; lower 8 bits in a1 (16 bit shift left). we then clear the top 8 bits of
; b1, and or in b1 with x1 in order to successfully swap order. this is
; then written back out.
;
; Arguments:        r0 - pointer to mp3 buffer
;                   r1 - length of mp3 buffer
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  process_mp3_buffer
process_mp3_buffer:                     ;assembly entry point
```

```

        ;save all registers we use so we can restore later
        PUSH_ACC a
        PUSH_ACC b
        move r0,ssh
        move x0,ssl

        nop
        do r1,process_buffer_loop
        nop

        move x:(r0),a ;we copy into a/b so we can do bitshifts.
        nop
        move a1,b

        lsl #16,a ;left-align the lower byte
        and #>$ff00,b ;select the top byte from b
        move b1,x0 ;move into x0 so that we can use or b1 with a
        or x0,a    ;we find our fixed word

        move a1,x:(r0)+ ;move our word back in

        nop ;spacing nops
        nop
        nop
process_buffer_loop:


        ;end of body code, now restore all registers
        move ssl,x0
        move ssh,r0
        POP_ACC b
        POP_ACC a
        nop
        rts     ;return from subroutine
;end function




; Fsetup_dac
; Functional Specification
;
; Description:      Fsetup_dac sets up the ESSI1 registers for the DAC's control
; interface, and sets it to reasonable values:
;   - 192KHz oversampling
;   - analog mute/infinite zero detect
;   - auto-ramp volume
;   - anti-clipping mode
;   - filter #2: minimum phase linear (like AMB's gamma-2 setting B)
;
; Operation:        Fsetup_dac writes the ESSI registers, then calls
; write_sync_dac to commit the commands.
;
; Arguments:        None.
;
```

```
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    None.
;
; Revision History: 2011/3/25    Raymond Jimenez    first revision
;
        global  Fsetup_dac
Fsetup_dac:                         ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        move    r0,ssh
        move    r1,ssl

        ;initialize the ESSI 1 control registers
        move    #>ESSI1_CRA,r0
        move    r0,x:M_CRA1
        move    #>ESSI1_CRB,r0
        move    r0,x:M_CRB1

        ;now configure the ESSI 1 pins as active and not GPIO
        move    #>ESSI1_PCRD,r0
        move    r0,x:M_PCRD
        move    #>ESSI1_PRRD,r0
        move    r0,x:M_PRRD

        ;we load first data
        move    #>$0,r0
        move    r0,x:M_TX10

```

```
        ;enable transmitter
        bset    #M_SSTE0,x:M_CRB1


        ;now that the ESSI port is setup, we can go ahead and write our initial
        ; reg values
        move    #>(DAC_RESET_ADDR),r0
        jsr     write_sync_dac          ;reset the chip to a known state


        move    #>(DAC_VOL_ADDR|DAC_VOL_CONTROL),r0     ;setup vol control
        jsr     write_sync_dac
        move    #>(DAC_FILTER_ADDR|DAC_FILTER_CONTROL),r0
                                                        ;setup filters
        jsr     write_sync_dac
        move    #>(DAC_MODE1_ADDR|DAC_MODE_CONTROL_1),r0
                                                        ;setup oversampling
        jsr     write_sync_dac

        move    #>INIT_ATTEN,r0
        move    r0,x:sound_atten

        ;set our initial volume
        jsr     Fvol_down
        jsr     Fvol_up

;end of body code, now restore all registers
        move    ssl,r1
        move    ssh,r0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Fsetup_dac


; write_sync_dac
; Functional Specification
;
; Description:      write_sync_dac writes a DAC register value to the
; DAC.
;
; ESSI1 interrupts must be disabled before invocation; they should never
; be enabled in the first place.
;
; Operation:        write_sync_dac simply waits on TDE to ensure that all
; data has been transmitted. it then moves the requested data and transmits
; it, and then returns. the magic of ESSI takes care of our frame sync.
;
; Arguments:        r0 - data to write, left aligned
```

```
;                         r0[23:17] - address of register to write
;                         r0[16] - should always be 0
;                         r0[15-8] - data to write
;                         r0[7-0] - reserved; write 0 for compat
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:    We require the strange bitfield structure of r0 to make
; things faster; since we mostly deal with pre-composed constants, we can use
; macro shifts to pre-calculate everything.
;
; Revision History: 2011/03/25    Raymond Jimenez    first revision
;
        global  write_sync_dac
write_sync_dac:                       ;assembly entry point

        ;save all registers we use so we can restore later

        ;we first make sure the transmitter is empty
        jclr    #M_TDE,x:M_SSISR1,*   ;keep on jumping here until TDE is set
        move    r0,x:M_TX10           ;transmit first 16 bits: command, address

        nop  ;respect the 3-cycle pipeline delay; we return when flags have been set
                nop
                nop

        ;restore all registers. we have.. none?

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine
;end write_sync_dac

; Fvol_up
; Functional Specification
```

```
;
; Description:      vol_up is called whenever we want to increase the volume. it
; increases it by ten -.125dB steps.
;
; Operation:        vol_up simply calls write_sync_dac in order to tell the dac
; go up to the desired volume.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/03/27    Raymond Jimenez    first revision
;
        global  Fvol_up
Fvol_up:                        ;C entry point

        ;standard C prologue
        move    #0,n6            ;k is the amount of local space
        move    ssh,x:(r6)+     ;save the return address
        move    (r6)+n6          ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        PUSH_ACC a
        move    x0,ssh
        move    r0,ssl


        move    x:sound_atten,a         ;we grab our current attenuation state
        move    #>ATTEN_STEP,x0         ;move step into x0 so we can subtract it
```

```
        sub     x0,a                    ; subtract (smaller atten == louder)

        cmp     #>MIN_ATTEN,a           ;see if it's too loud, nowhere to go
        jlt     vol_up_done             ;too loud, so go don't make any changes


        move    a1,x:sound_atten        ;store the change

        and     #>ATTEN_MASK,a          ;we now have our desired attenuation
                                        ; in a1, so we use masks to get it to
                                        ; the DAC.

        move    a1,x0                   ;keep a copy in x0, we'll be needing it
                                        ;(we do two 5-bit writes)
        and     #>ATTEN_REG_MASK,a      ;clear out to leave the bottom 5 bits

        ;we now transmit the lower 5 bits
        lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
        or      #>DAC_LLSB_ADDR,a       ;compose address

        move    a1,r0                   ;ready for calling
        jsr     write_sync_dac          ;effect the change

        ;now we transmit the high 5 bits, and latch the changes
        move    x0,a                    ;grab the old copy
        lsr     #5,a                    ;dac volume register is 5 bits wide per
        and     #>ATTEN_REG_MASK,a           ;clear out the top 5 bits
        lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
        or      #>(DAC_LMSB_ADDR|(ATTEN_LATCH_MASK<<DAC_DATA_SHIFT)),a
                                        ;compose address & commit change bit
        move    a1,r0
        jsr     write_sync_dac          ;really effect the change

vol_up_done:

;end of body code, now restore all registers
        move    ssl,r0
        move    ssh,x0
        POP_ACC a


        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Fvol_up


; Fvol_down
; Functional Specification
;
; Description:      vol_down is called whenever we want to increase the volume.
```

```
it
; increases it by ten -.125dB steps.
;
; Operation:        vol_down simply calls write_sync_dac in order to tell the dac
; go up to the desired volume.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/03/27    Raymond Jimenez    first revision
;
        global  Fvol_down
Fvol_down:                      ;C entry point

        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k


        ;save all registers we use so we can restore later
        PUSH_ACC a
        move    x0,ssh
        move    r0,ssl


        move    x:sound_atten,a         ;we grab our current attenuation state
        move    #>ATTEN_STEP,x0         ;move step into x0 so we can subtract it
        add     x0,a                    ; add (more atten == softer)

```

```
        cmp     #>MAX_ATTEN,a           ; see if it's too soft.
        jgt     vol_down_done            ;too soft, so go don't make any changes


        move    a1,x:sound_atten        ;store the change
        and     #>ATTEN_MASK,a          ;we now have our desired attenuation
                                        ; in a1, so we use masks to get it to
                                        ; the DAC.

        move    a1,x0                   ;keep a copy in x0, we'll be needing it
                                        ;(we do two 5-bit writes)
        and     #>ATTEN_REG_MASK,a           ;clear out the top 5 bits

        ;we now transmit the lower 5 bits
        lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
        or      #>DAC_LLSB_ADDR,a       ;compose address

        move    a1,r0                   ;ready for calling
        jsr     write_sync_dac          ;effect the change

        ;now we transmit the high 5 bits, and latch the changes
        move    x0,a                    ;grab the old copy
        lsr     #5,a                    ;dac volume register is 5 bits wide per
        and     #>ATTEN_REG_MASK,a           ;clear out the top 5 bits
        lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
        or      #>(DAC_LMSB_ADDR|(ATTEN_LATCH_MASK<<DAC_DATA_SHIFT)),a
                                        ;compose address & commit change bit
        move    a1,r0
        jsr     write_sync_dac          ;really effect the change

vol_down_done:

;end of body code, now restore all registers
        move    ssl,r0
        move    ssh,x0
        POP_ACC a


        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop     ;pipeline cache issue (see errata)
        rts     ;return from subroutine

;end Fvol_down




; get_atten
; Functional Specification
;
; Description:      This function returns the current attenuation
; status in steps of .125dB.
```

```
;
; Operation:        This function is an accessor function for sound_atten;
; it moves sound_atten into a.
;
; Arguments:        None.
;
; Return Values:    a1 - current attenuation in steps of .125dB
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  get_atten
get_atten:                      ;assembly entry point

        ;save all registers we use so we can restore later

        move    x:sound_atten,a

        ;end of body code, now restore all registers
        rts     ;return from subroutine
;end get_atten


        endsec



```

8.8 Timing

timing.inc contains definitions for timing, such as how many loop iterations correspond to a minimum of 1 millisecond, etc.

timing.asm provides several timing functions, such as Felapsed_time and others.

```
TICKS_PER_MS            EQU     1
SLEEP_CYCLES            EQU     8

```

```
;
; timing.asm
;
; This file offers several timing provider routines.
;

        section timing
 INCLUDE 'timing.inc'
 INCLUDE 'macros.inc'
        org     x:

 global time_elapsed_counter
time_elapsed_counter:
        ds      $1              ;one word to count the clock ticks elapsed.

        org     p:              ;we want to be located in program space

; Felapsed_time
; Functional Specification
;
; Description:      Felapsed_time provides the number of milliseconds it has
; been since it was last called.
;
; Operation:        It retrieves the value of the time_elapsed_counter, divides
; it by TICKS_PER_MS, and then returns this value.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Felapsed_time
```

```
Felapsed_time:              ;C entry point
        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

        ;save all registers we use so we can restore later
        move    r0,ssh
        move    x0,ssl

        move    #>0,r0

        ;we turn off interrupts for a bit to ensure we don't lose ticks
        ; there is no easy atomic alternative (no XCHG even!)
        START_CRITICAL
        move    x:time_elapsed_counter,a
        move    r0,x:time_elapsed_counter
        END_CRITICAL

        ;we rely on our handler being called every millisecond.

        ;end of body code, now restore all registers
        move    ssl,x0
        move    ssh,r0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        rts     ;return from subroutine
;end Felapsed_time

; time_elapsed_update
; Functional Specification
;
; Description:      time_elapsed_update should be called every
; millisecond. It updates a counter in memory, and it is the job
; of time_elapsed to properly interpret this counter's value.
;
; Operation:        It pulls in the counter into a register, increments
; it, and writes it back out to memory.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
```

```
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  time_elapsed_update
time_elapsed_update:                            ;assembly entry point

        ;save all registers we use so we can restore later
        move    r0,ssh                  ;we save r0 since we alter it


        move    x:time_elapsed_counter,r0 ;pull in our current counter
        move    (r0)+                   ;increment it
        move    r0,x:time_elapsed_counter ;then move it back


        ;end of body code, now restore all registers
        move    ssh,r0                  ;and restore r0 to original state.

        rts     ;return from subroutine
;end time_elapsed_update

; sleep
; Functional Specification
;
; Description:      sleep implements a sleep function using a delay loop.
; It is guaranteed to give a minimum of the sleep period, but the maximum
; is undefined (due to interrupts it is possible to sleep forever, beware.)
;
; Operation:        sleep simply loops the specified number of times, reading
; from an SRAM memory address to ensure the DSP can't optimize the loop away.
;
; Arguments:        r0 - time to sleep in multiples of 100ns
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
```

```
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  sleep
sleep:                      ;assembly entry point

        ;save all registers we use so we can restore later
        move    r1,ssh


        do      r0,outer_sleep_loop

        do      #SLEEP_CYCLES,inner_sleep_loop

        move    x:$0,r1         ;this instruction ensures at least one clock
                                ;period of wait, due to internal SRAM access

        ;spacing nops
        nop
        nop
        nop
inner_sleep_loop:

        ;spacing nops
        nop
        nop
        nop
outer_sleep_loop:


        ;end of body code, now restore all registers
        move    ssh,r1

        rts     ;return from subroutine
;end sleep


        endsec
```

8.9 Interrupts

intrrpts.asm contains the interrupt installation/initalivation code.

```
; intrruputs.asm
;
; This file offers interrupt setup and handler routines.
;
; void setup_timer0(void) -- sets up timer 0, currently used for processing
;   display commands at 500ns intervals.
;
; void install_timer0_handler(void) -- installs timer0 handler (currently calls
;   display function only)
;
; void setup_keypad_irq(void) -- sets up IRQ A, the keypad interrupt, and installs
;   the keypad handler
;

        section interrupts
 INCLUDE 'DSP_EQU.inc'
 INCLUDE 'display.inc'
 INCLUDE 'macros.inc'
        org     x:


        org     p:              ;we want to be located in program space

; Fstart_intr_setup
; Functional Specification
;
; Description:      Fstart_intr_setup should be called before setting
; up any interrupts. (it disables them).
;
; Operation:        It forces the SR interrupt mask pins to be 3, masking
; all interrupts.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
```

```
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fstart_intr_setup
Fstart_intr_setup:              ;C entry point
        ;standard C prologue
        move    #0,n6            ;k is the amount of local space
        move    ssh,x:(r6)+     ;save the return address
        move    (r6)+n6          ;allocate local stack space of size k

        ;save all registers we use so we can restore later (we don't use any)

        FORCE_INTERRUPTS_OFF

        ;end of body code, now restore all registers

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6          ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh       ;get return address
        nop
        rts     ;return from subroutine
;end Fstart_intr_setup


        global  Fsetup_timer0   ;prologue
Fsetup_timer0:                   ;C entry point

        move    #0,n6            ;k is the amount of local space
        move    ssh,x:(r6)+     ;save the return address
        move    (r6)+n6          ;allocate local stack space of size k

;Save all registers to be used.
        ;put a on stack, must keep it clean
        PUSH_ACC a
        move    x0,ssh


;Function body.

        ;we setup the interrupt handler before enabling the timer
        move    #>$0bf080,x0            ;unconditional branch to our subroutine
                                        ;(long interrupt)
        move    x0,p:I_TIM0C            ;install handler
        move    #>timer0_handler,x0
        move    x0,p:I_TIM0C+1          ;install address of our handler

        ;and remember to enable the interrupt handler globally
        ;(SR interrupt bits are already set by crt0)
        movep   x:M_IPRP,a
        and     #~M_T0L,a               ;clear the timer bits of the IPRP
        or      #(2<<M_T0L0),a          ;and force it to high (but
                                        ; maskable) priority
```

```
        movep   a1,x:M_IPRP             ;write back to the register

        ;setup the prescaler timer to have 4 cycles.
        ;@ 40MHz = 80/2 MHz, this gives us 1us resolution
        movep   #40,x:M_TPLR

        ;we want to have 200 cycles -> 200us command cycle time for the display
        movep   #200,x:M_TCPR0
        ;initialize the timer start count to zero, too
        movep   #0,x:M_TLR0

        ;set timer 0 to be enabled and in event counting mode, continuously
        ; counting and set it to trigger interrupts.
        movep   #(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR0

        ;then set the timer to enable.
        movep   #(1<<M_TE)|(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR0


;Restore all registers used.
        move    ssh,x0
        POP_ACC a

        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts


        global  Fsetup_timer1   ;prologue
Fsetup_timer1:                  ;C entry point

        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

;Save all registers to be used.
        ;put a on stack, must keep it clean
        PUSH_ACC a
        move    x0,ssh


;Function body.

        ;we setup the interrupt handler before enabling the timer
        move    #>$0bf080,x0            ;unconditional branch to our subroutine
                                        ;(long interrupt)
        move    x0,p:I_TIM1C            ;install handler
        move    #>timer1_handler,x0
        move    x0,p:I_TIM1C+1          ;install address of our handler

        ;and remember to enable the interrupt handler globally
        ;(SR interrupt bits are already set by crt0)
        ; this should have been setup already by timer0_setup.
```

```
        ; 1ms per tick.
        ; additionally, we base our MP3 timing off of this routine,
        ; down to millisecond accuracy, so don't change!
        movep   #1000,x:M_TCPR1
        ;initialize the timer start count to zero, too
        movep   #0,x:M_TLR1

        ;set timer 1 to be enabled and in event counting mode, continuously
        ; counting and set it to trigger interrupts.
        movep   #(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR1

        ;then set the timer to enable.
        movep   #(1<<M_TE)|(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR1


        ;we also setup the time_elapsed_counter which time_elapsed depends
        ; on:
        move    #>0,x0
        move    x0,x:time_elapsed_counter

;Restore all registers used.
        move    ssh,x0
        POP_ACC a

        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts




        global  Fsetup_timer2   ;prologue
Fsetup_timer2:                  ;C entry point

        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

;Save all registers to be used.
        ;put a on stack, must keep it clean
        PUSH_ACC a
        move    x0,ssh


;Function body.

        ;we setup the interrupt handler before enabling the timer
        move    #>$0bf080,x0            ;unconditional branch to our subroutine
                                        ;(long interrupt)
        move    x0,p:I_TIM2C            ;install handler
        move    #>timer2_handler,x0
        move    x0,p:I_TIM2C+1          ;install address of our handler

        ;and remember to enable the interrupt handler globally
```

```
        ;(SR interrupt bits are already set by crt0)
        ; this should have been setup already by timer0_setup.

        ; 15us per tick.
        ; dram refresh, so it's important that this works properly.
        movep   #400,x:M_TCPR2
        ;initialize the timer start count to zero, too
        movep   #0,x:M_TLR2

        ;set timer 2 to be enabled and in GPIO mode, continuously
        ; counting and set it to trigger interrupts. we latch off the
        ; internal clock/2 so that we have a 25ns pulse instead of some long
        ; thing.
        movep   #(1<<M_TCIE)|(0<<4)|(0<<M_PCE)|(1<<M_TRM)|(1<<M_DIR),x:M_TCSR2

        ;then set the timer to enable.
        movep   #(1<<M_TE)|(1<<M_TCIE)|(0<<4)|(0<<M_PCE)|(1<<M_TRM)|(1<<M_DIR),x:M_TCSR2

;Restore all registers used.
        move    ssh,x0
        POP_ACC a

        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address
        nop
        rts





        global  timer0_handler ;prologue
timer0_handler:                 ;interrupt entry point

        jsr     display_update ;call display update function

        rti     ;return; interrupts return with RTI



        global  timer1_handler ;prologue

timer1_handler:                 ;interrupt entry point

        jsr     time_elapsed_update ;call time_elapsed update function

        rti     ;return; interrupts return with RTI


timer2_handler:                 ;interrupt entry point

        ;how long does a refresh take? well, let's see, we need to hold the bus
        ; for about... 300ns? we'll hold for 400 just in case.
        ;each instruction is ~12.5ns, so that's 32 instructions.
```

```



        ; save registers we use
        move    r0,ssh
        move    r1,ssl
        move    r2,ssh
        move    #$200000,r0       ;we just grab the first 32 words of dram into r1.
                                  ; (it's important that it's dram due to DMA contention)
        ;we disable DMA during refresh.
        move    x:M_DCR2,r2
        bclr    #M_DE,x:M_DCR2                  ;disable the current DMA
;        move    x:(r0)+,r1
        nop
        nop
        nop
        jset    #M_DACT,x:M_DSTR,*              ;wait for DMA transfer to finish


        nop
        nop
        nop
        nop
        nop
        nop

       ;depend on BG to tell us when to stop reading.
        ; we are only returned control when the refresh is done.

        ;we first pulse our refresh pin:
        bset    #M_DO,x:M_TCSR2
        bclr    #M_DO,x:M_TCSR2

       ;depend on BG to tell us when to stop reading.
        ; we are only returned control when the refresh is done.
;        move    x:(r0)+,r1
;        move    x:(r0)+,r1

        nop
        move    r2,x:M_DCR2

        move    ssh,r2
        move    ssl,r1
        move    ssh,r0


        rti     ;return; interrupts return with RTI



; Fsetup_keypad_irq
; Functional Specification
;
; Description:      Fsetup_keypad_irq sets up the interrupts for
; the keypad.
```

```
;
; Operation:        It writes the interrupt pin setup registers
; (IPRC) to setup IRQA as a level-triggered pin with priority 1
; (low, we don't care about losing keys too much). We implement
; the interrupt handler with a fast handler (2 opcodes), that are
; copied from memory.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fsetup_keypad_irq
Fsetup_keypad_irq:                  ;assembly entry point
        ;standard C prologue
        move    #0,n6               ;k is the amount of local space
        move    ssh,x:(r6)+         ;save the return address
        move    (r6)+n6             ;allocate local stack space of size k

        ;save all registers we use so we can restore later
        move    x0,ssh              ;save x0
        PUSH_ACC a

        ;we first install the interrupt handler
        move    #>$0bf080,x0                ;unconditional branch to our subroutine
                                            ;(long interrupt)
        move    x0,p:I_IRQA                 ;install handler
        move    #keypad_handler,x0          ;and install handler address
        move    x0,p:I_IRQA+1

        ;then we setup the IRQ pin/IRQ registers
        move    x:M_IPRC,a          ;grab into a so we can operate on it
```

```
        and     #~M_IAL,a       ;clear the mode mask of IRQA
        or      #(1<<M_IAL2)|(1<<M_IAL0),a
                                ;we set it to be edge-triggered (M_IAL2 = 1)
                                ; and priority 1 (M_IAL[0-1] = 1
        move    a1,x:M_IPRC     ;and replace back

        ;end of body code, now restore all registers
        POP_ACC a
        move    ssh,x0          ;restore x0

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop
        rts     ;return from subroutine
;end Fsetup_keypad_irq


; Fend_intr_setup
; Functional Specification
;
; Description:      Fend_intr_setup should be called after setting up interrupts
; (it unmasks them).
;
; Operation:        It forces the SR interrupt mask pins to be 0, unmasking
; all interrupts.
;
; Arguments:        None.
;
; Return Values:    None.
;
; Global Variables: None.
;
; Shared Variables: None.
;
; Local Variables:  None.
;
; Inputs:           None.
;
; Outputs:          None.
;
; Error Handling:   None.
;
; Algorithm:        None.
;
; Data Structures:  None.
;
; Registers changed: None.
;
; Limitations:      None known.
;
; Known Bugs:       None.
;
; Special Notes:
```

```
;
; Revision History: 2011/02/16    Raymond Jimenez    first revision
;
        global  Fend_intr_setup
Fend_intr_setup:              ;C entry point
        ;standard C prologue
        move    #0,n6           ;k is the amount of local space
        move    ssh,x:(r6)+    ;save the return address
        move    (r6)+n6         ;allocate local stack space of size k

        ;save all registers we use so we can restore later (we don't use any)

        FORCE_INTERRUPTS_ON

        ;end of body code, now restore all registers

        ;standard C epilogue
        move    #0+1,n6
        move    (r6)-n6         ;deallocate local stack space, set ccr flags
        tst     a
        move    x:(r6),ssh      ;get return address

        nop
        rts     ;return from subroutine
;end Fend_intr_setup


        endsec
```

8.10 User Interface

Several portions of the provided code were modified in order to interface with the assembly code. They are provided as follows:

keyproc.h was updated to include volume-changing functions.

interfac.h was modified to reflect our current system, and was also updated to take into account the two volume up/down keys.

mainloop.c was modified to include volume-changing functions, as well as to call the various initialization functions.

keyupdat.c was modified to include volume-changing functions.

test_dram.c was a useful LFSR-based memory test utility.

```
/****************************************************************************/
/*                                                                          */
/*                              KEYPROC.H                                   */
/*                       Key Processing Functions                           */
/*                             Include File                                 */
/*                         MP3 Jukebox Project                              */
/*                              EE/CS  52                                   */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constants and function prototypes for the key
   processing functions defined in ffrev.c, keyupdat.c, and playmp3.c.


   Revision History:
      6/4/00   Glen George       Initial revision (from the 3/6/99 version of
                                 keyproc.h for the Digital Audio Recorder
                                 Project).
      6/5/08   Glen George       Added declarations for dec_FFRev_rate() and
                                 inc_FFRev_rate() functions.
      3/3/11   Raymond Jimenez   Added declarations for do_VolUp() and do_VolDown().
*/



#ifndef  I__KEYPROC_H__
    #define  I__KEYPROC_H__


/* library include files */
  /* none */

/* local include files */
#include  "mp3defs.h"




/* constants */
    /* none */




/* structures, unions, and typedefs */
    /* none */




/* function declarations */

enum status  no_action(enum status);      /* nothing to do */
enum status  stop_idle(enum status);      /* <Stop> when doing nothing */

enum status  do_TrackUp(enum status);     /* go to the next track */
```

```
enum status  do_TrackDown(enum status);   /* go to the previous track */

enum status  do_VolUp(enum status);       /* increase volume */
enum status  do_VolDown(enum status);     /* decrease volume */

enum status  start_Play(enum status);     /* begin playing the current track */
enum status  begin_Play(enum status);     /* start playing from fast forward or
reverse */
enum status  stop_Play(enum status);      /* stop playing */

enum status  start_RptPlay(enum status);  /* begin repeatedly playing the current
track */
enum status  cont_RptPlay(enum status);   /* switch to repeat play from standard play
*/
enum status  begin_RptPlay(enum status);  /* start repeatedly playing from fast
forward or reverse */

enum status  start_FastFwd(enum status);  /* start going fast forward */
enum status  switch_FastFwd(enum status); /* switch to fast forward from play */
enum status  begin_FastFwd(enum status);  /* switch to fast forward from reverse */

enum status  start_Reverse(enum status);  /* start going reverse */
enum status  switch_Reverse(enum status); /* switch to reverse from play */
enum status  begin_Reverse(enum status);  /* switch to reverse from fast forward */

enum status  stop_FFRev(enum status);     /* stop fast forward or reverse */

void         dec_FFRev_rate(void);        /* decrease fast forward/reverse speed */
void         inc_FFRev_rate(void);        /* increase fast forward/reverse speed */


#endif
```

```c
/****************************************************************************/
/*                                                                          */
/*                              INTERFAC.H                                  */
/*                         Interface Definitions                            */
/*                             Include File                                 */
/*                          MP3 Jukebox Project                             */
/*                               EE/CS 52                                   */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constants for interfacing between the C code and
   the assembly code/hardware.  This is a sample interface file to allow test
   compilation and linking of the code.


   Revision History:
      6/3/00   Glen George       Initial revision.
      4/2/01   Glen George       Removed definitions of DRAM_SIZE and
                             IDE_SIZE, they are no longer used.
      6/5/03   Glen George       Added constant definitions of TIME_NONE,
                             PARENT_DIR_CHAR, and SUBDIR_CHAR.
      4/29/06  Glen George       Updated value of IDE_BLOCK_SIZE to be in
                             units of words, not bytes.
*/



#ifndef  I__INTERFAC_H__
    #define  I__INTERFAC_H__


/* library include files */
  /* none */

/* local include files */
  /* none */



/* changed temporarily while we debug DRAM */
#define  DRAM_STARTSEG   0xa0800

#define  KEY_TRACKUP     0
#define  KEY_TRACKDOWN   1
#define  KEY_PLAY        2
#define  KEY_RPTPLAY     3
#define  KEY_FASTFWD     4
#define  KEY_REVERSE     5
#define  KEY_STOP        6
#define  KEY_VOLUP       7
#define  KEY_VOLDOWN     8
#define  KEY_ILLEGAL     9

#define  TIME_NONE       65535

#define  PARENT_DIR_CHAR '<'
```

```
#define   SUBDIR_CHAR      '>'

#define   STATUS_PLAY      0
#define   STATUS_FASTFWD   1
#define   STATUS_REVERSE   2
#define   STATUS_IDLE      3
#define   STATUS_ILLEGAL   4

#define   IDE_BLOCK_SIZE   256         /* 256 words/block */



#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                                MAINLOOP                                  */
/*                            Main Program Loop                             */
/*                           MP3 Jukebox Project                            */
/*                                EE/CS 52                                  */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the main processing loop (background) for the MP3
   Jukebox Project.  The only global function included is:
      main - background processing loop

   The local functions included are:
      key_lookup - get a key and look up its keycode

   The locally global variable definitions included are:
      none


   Revision History
      6/5/00   Glen George       Initial revision (from 3/6/99 version of
                                 mainloop.c for the Digital Audio Recorder
                                 Project).
      6/2/02   Glen George       Updated comments.
      5/15/03  Glen George       Moved static declarations to first keyword
                                 since the lame NIOS compiler requires it.
      5/15/03  Glen George       Changed type on some variables to size_t
                                 for better portability and more accurate
                 typing.
      5/15/03  Glen George       Added #include of stddef.h to get some
                                 standard definitions.
      6/5/03   Glen George       Removed references to track number, it is no
                             longer used.
      6/5/03   Glen George       Added initialization to FAT directory system.
      6/5/03   Glen George       Updated function headers.
*/



/* library include files */
#include  <stddef.h>

/* local include files */
#include  "interfac.h"
#include  "mp3defs.h"
#include  "keyproc.h"
#include  "updatfnc.h"
#include  "trakutil.h"
#include  "fatutil.h"




/* local function declarations */
enum keycode  key_lookup(void);      /* translate key values into keycodes */
```

```
/*
   main

   Description:      This procedure is the main program loop for the MP3
                     Jukebox.  It loops getting keys from the keypad,
                     processing those keys as is appropriate.  It also handles
                     updating the display and setting up the buffers for MP3
                     playback.

   Arguments:        None.
   Return Value:     (int) - return code, always 0 (never returns).

   Input:            Keys from the keypad.
   Output:           Status information to the display.

   Error Handling:   Invalid input is ignored.

   Algorithms:       The function is table-driven.  The processing routines
                     for each input are given in tables which are selected
                     based on the context (state) in which the program is
                     operating.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    June 5, 2003

*/

extern void test_dram();

int  main()
{
    /* variables */
    enum keycode  key;                      /* an input key */

    enum status   cur_status = STAT_IDLE;   /* current program status */
    enum status   prev_status = STAT_IDLE;  /* previous program status */

    long int      root_dir_start;       /* start of the root directory */

    /* array of status type translations (from enum status to #defines) */
    /* note: the array must match the enum definition order exactly */
    static const unsigned int  xlat_stat[] =
        {  STATUS_IDLE,      /* system idle */
           STATUS_PLAY,      /* playing (or repeat playing) a track */
           STATUS_FASTFWD,   /* fast forwarding a track */
           STATUS_REVERSE    /* reversing a track */
        };

    /* update functions (one for each system status type) */
    static enum status  (* const update_fnc[NUM_STATUS])(enum status) =
        /*                        Current System Status                           */
```

```
        /*    idle         play      fast forward      reverse      */
        {  no_update, update_Play, update_FastFwd, update_Reverse  };

    /* key processing functions (one for each system status type and key) */
    static enum status  (* const process_key[NUM_KEYCODES][NUM_STATUS])(enum status)
    =
        /*                            Current System Status
        */
        /* idle            play            fast forward   reverse                 key
        */
      { {  do_TrackUp,    no_action,      no_action,     no_action      },   /* <Track
      Up>     */
        {  do_TrackDown,  no_action,      no_action,     no_action      },   /* <Track
        Down>   */
        {  start_Play,    no_action,      begin_Play,    begin_Play     },   /* <Play>
        */
        {  start_RptPlay, cont_RptPlay,   begin_RptPlay, begin_RptPlay },   /*
        <Repeat Play>  */
        {  start_FastFwd, switch_FastFwd, stop_FFRev,    begin_FastFwd },   /* <Fast
        Forward> */
        {  start_Reverse, switch_Reverse, begin_Reverse, stop_FFRev     },   /*
        <Reverse>      */
        {  stop_idle,     stop_Play,      stop_FFRev,    stop_FFRev     },   /* <Stop>
        */
        {  do_VolUp,      do_VolUp,       do_VolUp,      do_VolUp       },   /*
        <Volume Up>    */
        {  do_VolDown,    do_VolDown,     do_VolDown,    do_VolDown     },   /*
        <Volume Down>  */
        {  no_action,     no_action,      no_action,     no_action      } }; /*
        illegal key    */

    /* startup/initialization routines */

    start_intr_setup();

    display_init();



    setup_timer0();

    setup_timer1();
    setup_timer2();   /* enable DRAM refresh timer. until interrupts are enabled,
                       DRAM is not useful. */

    setup_keypad_irq();

    key_init();

    setup_dac();

    setup_mp3();

    end_intr_setup();

    setup_ide();
/*
```

```c
    while(1)
    {
      test_dram();
    }
*/

    /* first initialize everything */
    /* initalize FAT directory functions */
    root_dir_start = init_FAT_system();

    /* get the first directory entry (file/song) */
    if (root_dir_start != 0)  {
    /* have a valid starting sector - get the first directory entry */
    get_first_dir_entry(root_dir_start);
    /* and setup the information for the track/file */
    setup_cur_track_info();
    }
    else  {
        /* had an error - fill with error track information */
    setup_error_track_info();
    }


    /* display track information */
    display_time(get_track_time());
    display_title(get_track_title());
    display_artist(get_track_artist());

    display_status(xlat_stat[cur_status]);  /* display status */
    display_volume(); /* the volume is also a part of the status */


    /* infinite loop processing input */
    while(TRUE)  {

        /* handle updates */
        cur_status = update_fnc[cur_status](cur_status);


        /* now check for keypad input */
        if (key_available())  {

            /* have keypad input - get the key */
            key = key_lookup();

            /* execute processing routine for that key */
            cur_status = process_key[key][cur_status](cur_status);
        }


        /* finally, if the status has changed - display the new status */
        if (cur_status != prev_status)  {

            /* status has changed - update the status display */
            display_status(xlat_stat[cur_status]);
            display_volume(); /* the volume is also a part of the status */
        }
```

```
        /* always remember the current status for next loop iteration */
        prev_status = cur_status;
    }


    /* done with main (never should get here), return 0 */
    return  0;

}




/*
   key_lookup

   Description:      This function gets a key from the keypad and translates
                     the raw keycode to an enumerated keycode for the main
                     loop.

   Arguments:        None.
   Return Value:     (enum keycode) - type of the key input on keypad.

   Input:            Keys from the keypad.
   Output:           None.

   Error Handling:   Invalid keys are returned as KEYCODE_ILLEGAL.

   Algorithms:       The function uses an array to lookup the key types.
   Data Structures:  Array of key types versus key codes.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    May 15, 2003

*/

static  enum keycode  key_lookup()
{
    /* variables */

    static const enum keycode  keycodes[] = /* array of keycodes */
        {                                   /* order must match keys array exactly */
           KEYCODE_TRACKUP,    /* <Track Up>     */    /* also needs to have extra */
           KEYCODE_TRACKDOWN,  /* <Track Down>   */    /* entry for illegal codes */
           KEYCODE_PLAY,       /* <Play>         */
           KEYCODE_RPTPLAY,    /* <Repeat Play>  */
           KEYCODE_FASTFWD,    /* <Fast Forward> */
           KEYCODE_REVERSE,    /* <Reverse>      */
           KEYCODE_STOP,       /* <Stop>         */
           KEYCODE_VOLUP,      /* <Volume Up>    */
           KEYCODE_VOLDOWN,    /* <Volume Down>  */
           KEYCODE_ILLEGAL     /* other keys     */
        };
```

```
    static const int  keys[] =   /* array of key values */
        {                        /* order must match keycodes array exactly */
           KEY_TRACKUP,    /* <Track Up>     */
           KEY_TRACKDOWN,  /* <Track Down>   */
           KEY_PLAY,       /* <Play>         */
           KEY_RPTPLAY,    /* <Repeat Play>  */
           KEY_FASTFWD,    /* <Fast Forward> */
           KEY_REVERSE,    /* <Reverse>      */
           KEY_STOP,       /* <Stop>         */
           KEY_VOLUP,      /* <Volume Up>    */
           KEY_VOLDOWN     /* <Volume Down>  */
        };

    int     key;           /* an input key */

    size_t  i;             /* general loop index */



    /* get a key */
    key = getkey();


    /* lookup key in keys array */
    for (i = 0; ((i < (sizeof(keys)/sizeof(int))) && (key != keys[i])); i++);


    /* return the appropriate key type */
    return  keycodes[i];

}
```

```
/****************************************************************************/
/*                                                                          */
/*                                KEYUPDAT                                  */
/*             Miscellaneous Key Processing and Update Functions            */
/*                           MP3 Jukebox Project                            */
/*                                EE/CS  52                                 */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the key processing and update functions for operations
   other than Play, Record, Fast Forward, and Reverse for the MP3 Jukebox
   Project.  These functions are called by the main loop of the MP3 Jukebox.
   The functions included are:
      do_TrackUp      - go to the next track (key processing function)
      do_TrackDown    - go to the previous track (key processing function)
      no_action       - nothing to do (key processing function)
      no_update       - nothing to do (update function)
      stop_idle       - stop when doing nothing (key processing function)
      do_VolUp        - increase volume (key processing function)
      do_VolDown      - decrease volume (key processing function)

   The local functions included are:
      none

   The global variable definitions included are:
      none


   Revision History
      6/6/00   Glen George       Initial revision (from 3/6/99 version of
                                 keyupdat.c for the Digital Audio Recorder
                                 Project).
      6/2/02   Glen George       Updated comments.
      6/5/03   Glen George       Updated do_TrackUp and do_TrackDown to handle
                                 directory traversal in order to support FAT
                                 directory structures, they now go up and down
                                 in the current directory.
      6/5/03   Glen George       Added #include of fatutil.h for function
                                 declarations needed by above change.
      6/5/03   Glen George       Updated function headers.
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "mp3defs.h"
#include  "keyproc.h"
#include  "updatfnc.h"
#include  "trakutil.h"
#include  "fatutil.h"
```

```
/*
   no_action

   Description:      This function handles a key when there is nothing to be
                     done.  It just returns.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 5, 1994

*/

enum status  no_action(enum status cur_status)
{
    /* variables */
      /* none */



    /* return the current status */
    return  cur_status;

}




/*
   do_TrackUp

   Description:      This function handles the <Track Up> key when nothing is
                     happening in the system.  It moves to the previous entry
                     in the directory and resets the track time and loads the
                     track information for the new track.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           The new track information is output.

   Error Handling:   None.
```

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    June 5, 2003

*/

enum status  do_TrackUp(enum status cur_status)
{
    /* variables */
      /* none */



    /* move to the previous directory entry, watching for errors */
    if (!get_previous_dir_entry())
        /* successfully got the new entry, load its data */
        setup_cur_track_info();
    else
        /* there was an error - load error track information */
        setup_error_track_info();


    /* display the track information for this track */
    display_time(get_track_time());
    display_title(get_track_title());
    display_artist(get_track_artist());


    /* done so return the current status */
    return  cur_status;

}




/*
   do_TrackDown

   Description:      This function handles the <Track Down> key when nothing
                     is happening in the system.  It moves to the next entry
                     in the directory and resets the track time and loads the
                     track information for the new track.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           The new track information is output.

   Error Handling:   None.

   Algorithms:       None.
```

```
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    June 5, 2003

*/

enum status  do_TrackDown(enum status cur_status)
{
    /* variables */
      /* none */



    /* move to the next directory entry, watching for errors */
    if (!get_next_dir_entry())
        /* successfully got the new entry, load its data */
        setup_cur_track_info();
    else
        /* there was an error - load error track information */
        setup_error_track_info();


    /* display the track information for this track */
    display_time(get_track_time());
    display_title(get_track_title());
    display_artist(get_track_artist());


    /* done so return the current status */
    return  cur_status;

}




/*
   stop_idle

   Description:      This function handles the <Stop> key when nothing is
                     happening in the system.  It just resets the track time
                     and variables to indicate the start of the track.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           The new track time (the track length) is output.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.
```

```
   Shared Variables: None.

   Author:           Glen George
   Last Modified:    June 3, 2000

*/

enum status  stop_idle(enum status cur_status)
{
    /* variables */
      /* none */



    /* reset to the start of the current track */
    init_track();

    /* display the new time for the current track */
    display_time(get_track_time());


    /* return with the status unchanged */
    return  cur_status;

}




/*
   no_update

   Description:      This function handles updates when there is nothing to
                     do.  It just returns with the status unchanged.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    Mar. 5, 1994

*/

enum status  no_update(enum status cur_status)
{
    /* variables */
      /* none */
```

```
    /* nothing to do - return with the status unchanged */
    return  cur_status;

}


/*
   do_VolUp

   Description:      This function handles when a key is pressed to increase
volume. It asks the corresponding assembly function, vol_up, to increase the
volume.

   Arguments:        cur_status (enum status) - the current system status.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Raymond Jimenez
   Last Modified:    Mar. 27, 2011

*/

enum status  do_VolUp(enum status cur_status)
{
    /* variables */
      /* none */

    vol_up(); /* vol_up doesn't take any arguments; it does the bounds checking */

    display_volume(); /* and reflect the change in our UI */

    /* nothing to do - return with the status unchanged */
    return  cur_status;

}


/*
   do_VolDown

   Description:      This function handles when a key is pressed to increase
volume. Much like do_VolUp, it asks its corresponding assembly function,
vol_down, to decrease the volume.

   Arguments:        cur_status (enum status) - the current system status.
```

```
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Raymond Jimenez
   Last Modified:    Mar. 27, 2011

*/

enum status  do_VolDown(enum status cur_status)
{
    /* variables */
      /* none */

    vol_down(); /* vol_down doesn't take any arguments; it does the bounds checking
    */

    display_volume(); /* and reflect the change in our UI */

    /* nothing to do - return with the status unchanged */
    return  cur_status;

}
```

```
#include <stddef.h>
#include <stdio.h>

#include "interfac.h"
#include "mp3defs.h"

#define BLOCK_SZ 0x2000
#define DRAM_SZ 0x4000
#define DRAM_START 0x200000

void test_dram()
{
  unsigned int lfsr = 1;
  unsigned int period = 0;
  volatile unsigned int *DRAMptr;
  char buffer[BLOCK_SZ];
  char strbuf[80];

  for(DRAMptr = (unsigned int *)(DRAM_START);\
      DRAMptr < (unsigned int *)(DRAM_START + DRAM_SZ);\
      DRAMptr += BLOCK_SZ)
  {

  lfsr = (unsigned int)DRAMptr;
  for(period = 0; period < BLOCK_SZ; period++)
  {
     /* taps at 24,23,22,17 */
    lfsr = (lfsr >> 1) ^ (-(lfsr & 0x1u) & 0xE10000u);
    DRAMptr[period] = lfsr;
  }

  for(period = 0; period < BLOCK_SZ; period++)
  {
    buffer[period] = 0xf10a7;
    buffer[period] = DRAMptr[period];
  }

  lfsr = (unsigned int)DRAMptr;
  for(period = 0; period < BLOCK_SZ; period++)
  {
    lfsr = (lfsr >> 1) ^ (-(lfsr & 0x1u) & 0xE10000u);
    if(buffer[period] == lfsr)
    {
      sprintf(&strbuf[0], "OK: %x", DRAMptr+period);
      display_artist(&strbuf[0]);
    } else {
      sprintf(&strbuf[0], "F: %x, %x, %x", DRAMptr+period,\
                                              buffer[period],\
                                              lfsr);
      display_title(&strbuf[0]);
    }

  }

  }
  return;
}
```

8.11 Miscellaneous

DSP_EQU.inc contains several system defines, such as critical memory addresses and register values.

general.inc contains important definitons applicable to all code.

macros.inc contains several important macros, such as a x86 PUSH/POP equivalent and critical code wrappers.

Makefile allows us to build and package.

```
;***********************************************************************************
;
; EQUATES for 56302/3 I/O registers and ports
;
; Last update: June 11 1995
;
;***********************************************************************************
    page 132,55,0,0,0
    opt mex

ioequ ident 1,0

;--------------------------------------------------------------------------
;
;   EQUATES for I/O Port Programming
;
;--------------------------------------------------------------------------

; Register Addresses
M_HDR       EQU $FFFFC9         ; Port B (host) GPIO data Register
M_HDDR      EQU $FFFFC8         ; Port B (host) GPIO direction Register
M_PCRC      EQU $FFFFBF         ; Port C (ESSI0) Control Register
M_PRRC      EQU $FFFFBE         ; Port C (ESSI0) Direction Register
M_PDRC      EQU $FFFFBD         ; Port C (ESSI0) GPIO Data Register
M_PCRD      EQU $FFFFAF         ; Port D (ESSI1) Control register
M_PRRD      EQU $FFFFAE         ; Port D (ESSI1) Direction Data Register
M_PDRD      EQU $FFFFAD         ; Port D (ESSI1) GPIO Data Register
M_PCRE      EQU $FFFF9F         ; Port E (SCI) Control register
M_PRRE      EQU $FFFF9E         ; Port E (SCI) Direction Register
M_PDRE      EQU $FFFF9D         ; Port E (SCI) Data Register
M_OGDB      EQU $FFFFFC         ; OnCE GDB Register


;--------------------------------------------------------------------------
;
;   EQUATES for Host Interface
;
;--------------------------------------------------------------------------

; Register Addresses
M_HCR       EQU $FFFFC2         ; Host Control Register
M_HSR       EQU $FFFFC3         ; Host Status Rgister
M_HPCR      EQU $FFFFC4         ; Host Polarity Control Register
M_HBAR      EQU $FFFFC5         ; Host Base Address Register
M_HRX       EQU $FFFFC6         ; Host Receive Register
M_HTX       EQU $FFFFC7         ; Host Transmit Register

; HCR bits definition
M_HRIE      EQU $0              ; Host Receive interrupts Enable
M_HTIE      EQU $1              ; Host Transmit Interrupt Enable
M_HCIE      EQU $2              ; Host Command Interrupt Enable
M_HF2       EQU $3              ; Host Flag 2
M_HF3       EQU $4              ; Host Flag 3

; HSR bits definition
```

```
M_HRDF      EQU $0              ; Host Receive Data Full
M_HTDE      EQU $1              ; Host Receive Data Emptiy
M_HCP       EQU $2              ; Host Command Pending
M_HF0       EQU $3              ; Host Flag 0
M_HF1       EQU $4              ; Host Flag 1

; HPCR bits definition
M_HGEN      EQU $0              ; Host Port GPIO Enable
M_HA8EN     EQU $1              ; Host Address 8 Enable
M_HA9EN     EQU $2              ; Host Address 9 Enable
M_HCSEN     EQU $3              ; Host Chip Select Enable
M_HREN      EQU $4              ; Host Request Enable
M_HAEN      EQU $5              ; Host Acknowledge Enable
M_HEN       EQU $6              ; Host Enable
M_HOD       EQU $8              ; Host Request Open Drain mode
M_HDSP      EQU $9              ; Host Data Strobe Polarity
M_HASP      EQU $A              ; Host Address Strobe Polarity
M_HMUX      EQU $B              ; Host Multiplexed bus select
M_HD_HS     EQU $C              ; Host Double/Single Strobe select
M_HCSP      EQU $D              ; Host Chip Select Polarity
M_HRP       EQU $E              ; Host Request PolarityPolarity
M_HAP       EQU $F              ; Host Acknowledge Polarity


;------------------------------------------------------------------------
;
;   EQUATES for Serial Communications Interface (SCI)
;
;------------------------------------------------------------------------

; Register Addresses
M_STXH      EQU $FFFF97         ; SCI Transmit Data Register (high)
M_STXM      EQU $FFFF96         ; SCI Transmit Data Register (middle)
M_STXL      EQU $FFFF95         ; SCI Transmit Data Register (low)
M_SRXH      EQU $FFFF9A         ; SCI Receive Data Register (high)
M_SRXM      EQU $FFFF99         ; SCI Receive Data Register (middle)
M_SRXL      EQU $FFFF98         ; SCI Receive Data Register (low)
M_STXA      EQU $FFFF94         ; SCI Transmit Address Register
M_SCR       EQU $FFFF9C         ; SCI Control Register
M_SSR       EQU $FFFF93         ; SCI Status Register
M_SCCR      EQU $FFFF9B         ; SCI Clock Control Register

; SCI Control Register Bit Flags
M_WDS       EQU $7              ; Word Select Mask (WDS0-WDS3)
M_WDS0      EQU 0               ; Word Select 0
M_WDS1      EQU 1               ; Word Select 1
M_WDS2      EQU 2               ; Word Select 2
M_SSFTD     EQU 3               ; SCI Shift Direction
M_SBK       EQU 4               ; Send Break
M_WAKE      EQU 5               ; Wakeup Mode Select
M_RWU       EQU 6               ; Receiver Wakeup Enable
M_WOMS      EQU 7               ; Wired-OR Mode Select
M_SCRE      EQU 8               ; SCI Receiver Enable
M_SCTE      EQU 9               ; SCI Transmitter Enable
M_ILIE      EQU 10              ; Idle Line Interrupt Enable
M_SCRIE     EQU 11              ; SCI Receive Interrupt Enable
M_SCTIE     EQU 12              ; SCI Transmit Interrupt Enable
```

```
M_TMIE      EQU 13              ; Timer Interrupt Enable
M_TIR       EQU 14              ; Timer Interrupt Rate
M_SCKP      EQU 15              ; SCI Clock Polarity
M_REIE      EQU 16              ; SCI Error Interrupt Enable (REIE)

; SCI Status Register Bit Flags
M_TRNE      EQU 0               ; Transmitter Empty
M_TDRE      EQU 1               ; Transmit Data Register Empty
M_RDRF      EQU 2               ; Receive Data Register Full
M_IDLE      EQU 3               ; Idle Line Flag
M_OR        EQU 4               ; Overrun Error Flag
M_PE        EQU 5               ; Parity Error
M_FE        EQU 6               ; Framing Error Flag
M_R8        EQU 7               ; Received Bit 8 (R8) Address

; SCI Clock Control Register
M_CD        EQU $FFF            ; Clock Divider Mask (CD0-CD11)
M_COD       EQU 12              ; Clock Out Divider
M_SCP       EQU 13              ; Clock Prescaler
M_RCM       EQU 14              ; Receive Clock Mode Source Bit
M_TCM       EQU 15              ; Transmit Clock Source Bit


;------------------------------------------------------------------------
;
;   EQUATES for Enhanced Synchronous Serial Interface (ESSI)
;
;------------------------------------------------------------------------
;
; Register Addresses Of ESSI0
M_TX00      EQU $FFFFBC         ; ESSI0 Transmit Data Register 0
M_TX01      EQU $FFFFBB         ; ESSIO Transmit Data Register 1
M_TX02      EQU $FFFFBA         ; ESSIO Transmit Data Register 2
M_TSR0      EQU $FFFFB9         ; ESSI0 Time Slot Register
M_RX0       EQU $FFFFB8         ; ESSI0 Receive Data Register
M_SSISR0    EQU $FFFFB7         ; ESSI0 Status Register
M_CRB0      EQU $FFFFB6         ; ESSI0 Control Register B
M_CRA0      EQU $FFFFB5         ; ESSI0 Control Register A
M_TSMA0     EQU $FFFFB4         ; ESSI0 Transmit Slot Mask Register A
M_TSMB0     EQU $FFFFB3         ; ESSI0 Transmit Slot Mask Register B
M_RSMA0     EQU $FFFFB2         ; ESSI0 Receive Slot Mask Register A
M_RSMB0     EQU $FFFFB1         ; ESSI0 Receive Slot Mask Register B

; Register Addresses Of ESSI1
M_TX10      EQU $FFFFAC         ; ESSI1 Transmit Data Register 0
M_TX11      EQU $FFFFAB         ; ESSI1 Transmit Data Register 1
M_TX12      EQU $FFFFAA         ; ESSI1 Transmit Data Register 2
M_TSR1      EQU $FFFFA9         ; ESSI1 Time Slot Register
M_RX1       EQU $FFFFA8         ; ESSI1 Receive Data Register
M_SSISR1    EQU $FFFFA7         ; ESSI1 Status Register
M_CRB1      EQU $FFFFA6         ; ESSI1 Control Register B
M_CRA1      EQU $FFFFA5         ; ESSI1 Control Register A
M_TSMA1     EQU $FFFFA4         ; ESSI1 Transmit Slot Mask Register A
M_TSMB1     EQU $FFFFA3         ; ESSI1 Transmit Slot Mask Register B
M_RSMA1     EQU $FFFFA2         ; ESSI1 Receive Slot Mask Register A
M_RSMB1     EQU $FFFFA1         ; ESSI1 Receive Slot Mask Register B

```

```
; ESSI Control Register A Bit Flags
M_PM        EQU $FF              ; Prescale Modulus Select Mask (PM0-PM7)
M_PSR       EQU 11               ; Prescaler Range
M_DC        EQU $1F000           ; Frame Rate Divider Control Mask (DC0-DC7)
M_ALC       EQU 18               ; Alignment Control (ALC)
M_WL        EQU $380000          ; Word Length Control Mask (WL0-WL7)
M_SSC1      EQU 22               ; Select SC1 as TR #0 drive enable (SSC1)
; ESSI Control Register B Bit Flags
M_OF        EQU $3               ; Serial Output Flag Mask
M_OF0       EQU 0                ; Serial Output Flag 0
M_OF1       EQU 1                ; Serial Output Flag 1
M_SCD       EQU $1C              ; Serial Control Direction Mask
M_SCD0      EQU 2                ; Serial Control 0 Direction
M_SCD1      EQU 3                ; Serial Control 1 Direction
M_SCD2      EQU 4                ; Serial Control 2 Direction
M_SCKD      EQU 5                ; Clock Source Direction
M_SHFD      EQU 6                ; Shift Direction
M_FSL       EQU $180             ; Frame Sync Length Mask (FSL0-FSL1)
M_FSL0      EQU 7                ; Frame Sync Length 0
M_FSL1      EQU 8                ; Frame Sync Length 1
M_FSR       EQU 9                ; Frame Sync Relative Timing
M_FSP       EQU 10               ; Frame Sync Polarity
M_CKP       EQU 11               ; Clock Polarity
M_SYN       EQU 12               ; Sync/Async Control
M_MOD       EQU 13               ; ESSI Mode Select
M_SSTE      EQU $1C000           ; ESSI Transmit enable Mask
M_SSTE2     EQU 14               ; ESSI Transmit #2 Enable
M_SSTE1     EQU 15               ; ESSI Transmit #1 Enable
M_SSTE0     EQU 16               ; ESSI Transmit #0 Enable
M_SSRE      EQU 17               ; ESSI Receive Enable
M_SSTIE     EQU 18               ; ESSI Transmit Interrupt Enable
M_SSRIE     EQU 19               ; ESSI Receive Interrupt Enable
M_STLIE     EQU 20               ; ESSI Transmit Last Slot Interrupt Enable
M_SRLIE     EQU 21               ; ESSI Receive Last Slot Interrupt Enable
M_STEIE     EQU 22               ; ESSI Transmit Error Interrupt Enable
M_SREIE     EQU 23               ; ESSI Receive Error Interrupt Enable
; ESSI Status Register Bit Flags
M_IF        EQU $3               ; Serial Input Flag Mask
M_IF0       EQU 0                ; Serial Input Flag 0
M_IF1       EQU 1                ; Serial Input Flag 1
M_TFS       EQU 2                ; Transmit Frame Sync Flag
M_RFS       EQU 3                ; Receive Frame Sync Flag
M_TUE       EQU 4                ; Transmitter Underrun Error FLag
M_ROE       EQU 5                ; Receiver Overrun Error Flag
M_TDE       EQU 6                ; Transmit Data Register Empty
M_RDF       EQU 7                ; Receive Data Register Full
; ESSI Transmit Slot Mask Register A
M_SSTSA     EQU $FFFF            ; ESSI Transmit Slot Bits Mask A (TS0-TS15)
; ESSI Transmit Slot Mask Register B
M_SSTSB     EQU $FFFF            ; ESSI Transmit Slot Bits Mask B (TS16-TS31)
; ESSI Receive Slot Mask Register A
M_SSRSA     EQU $FFFF            ; ESSI Receive Slot Bits Mask A (RS0-RS15)
```

```
; ESSI Receive Slot Mask Register B
M_SSRSB     EQU $FFFF           ; ESSI Receive Slot Bits Mask B (RS16-RS31)


;------------------------------------------------------------------------
;
;   EQUATES for Exception Processing
;
;------------------------------------------------------------------------

; Register Addresses
M_IPRC      EQU $FFFFFF         ; Interrupt Priority Register Core
M_IPRP      EQU $FFFFFE         ; Interrupt Priority Register Peripheral

; Interrupt Priority Register Core (IPRC)
M_IAL       EQU $7              ; IRQA Mode Mask
M_IAL0      EQU 0               ; IRQA Mode Interrupt Priority Level (low)
M_IAL1      EQU 1               ; IRQA Mode Interrupt Priority Level (high)
M_IAL2      EQU 2               ; IRQA Mode Trigger Mode
M_IBL       EQU $38             ; IRQB Mode Mask
M_IBL0      EQU 3               ; IRQB Mode Interrupt Priority Level (low)
M_IBL1      EQU 4               ; IRQB Mode Interrupt Priority Level (high)
M_IBL2      EQU 5               ; IRQB Mode Trigger Mode
M_ICL       EQU $1C0            ; IRQC Mode Mask
M_ICL0      EQU 6               ; IRQC Mode Interrupt Priority Level (low)
M_ICL1      EQU 7               ; IRQC Mode Interrupt Priority Level (high)
M_ICL2      EQU 8               ; IRQC Mode Trigger Mode
M_IDL       EQU $E00            ; IRQD Mode Mask
M_IDL0      EQU 9               ; IRQD Mode Interrupt Priority Level (low)
M_IDL1      EQU 10              ; IRQD Mode Interrupt Priority Level (high)
M_IDL2      EQU 11              ; IRQD Mode Trigger Mode
M_D0L       EQU $3000           ; DMA0 Interrupt priority Level Mask
M_D0L0      EQU 12              ; DMA0 Interrupt Priority Level (low)
M_D0L1      EQU 13              ; DMA0 Interrupt Priority Level (high)
M_D1L       EQU $C000           ; DMA1 Interrupt Priority Level Mask
M_D1L0      EQU 14              ; DMA1 Interrupt Priority Level (low)
M_D1L1      EQU 15              ; DMA1 Interrupt Priority Level (high)
M_D2L       EQU $30000          ; DMA2 Interrupt priority Level Mask
M_D2L0      EQU 16              ; DMA2 Interrupt Priority Level (low)
M_D2L1      EQU 17              ; DMA2 Interrupt Priority Level (high)
M_D3L       EQU $C0000          ; DMA3 Interrupt Priority Level Mask
M_D3L0      EQU 18              ; DMA3 Interrupt Priority Level (low)
M_D3L1      EQU 19              ; DMA3 Interrupt Priority Level (high)
M_D4L       EQU $300000         ; DMA4 Interrupt priority Level Mask
M_D4L0      EQU 20              ; DMA4 Interrupt Priority Level (low)
M_D4L1      EQU 21              ; DMA4 Interrupt Priority Level (high)
M_D5L       EQU $C00000         ; DMA5 Interrupt priority Level Mask
M_D5L0      EQU 22              ; DMA5 Interrupt Priority Level (low)
M_D5L1      EQU 23              ; DMA5 Interrupt Priority Level (high)

; Interrupt Priority Register Peripheral (IPRP)
M_HPL       EQU $3              ; Host Interrupt Priority Level Mask
M_HPL0      EQU 0               ; Host Interrupt Priority Level (low)
M_HPL1      EQU 1               ; Host Interrupt Priority Level (high)
M_S0L       EQU $C              ; ESSI0 Interrupt Priority Level Mask
M_S0L0      EQU 2               ; ESSI0 Interrupt Priority Level (low)
```

```
M_S0L1      EQU 3                   ; ESSI0 Interrupt Priority Level (high)
M_S1L       EQU $30                 ; ESSI1 Interrupt Priority Level Mask
M_S1L0      EQU 4                   ; ESSI1 Interrupt Priority Level (low)
M_S1L1      EQU 5                   ; ESSI1 Interrupt Priority Level (high)
M_SCL       EQU $C0                 ; SCI Interrupt Priority Level Mask
M_SCL0      EQU 6                   ; SCI Interrupt Priority Level (low)
M_SCL1      EQU 7                   ; SCI Interrupt Priority Level (high)
M_T0L       EQU $300                ; TIMER Interrupt Priority Level Mask
M_T0L0      EQU 8                   ; TIMER Interrupt Priority Level (low)
M_T0L1      EQU 9                   ; TIMER Interrupt Priority Level (high)


;------------------------------------------------------------------------
;
;   EQUATES for TIMER
;
;------------------------------------------------------------------------

; Register Addresses Of TIMER0
M_TCSR0     EQU $FFFF8F             ; TIMER0 Control/Status Register
M_TLR0      EQU $FFFF8E             ; TIMER0 Load Reg
M_TCPR0     EQU $FFFF8D             ; TIMER0 Compare Register
M_TCR0      EQU $FFFF8C             ; TIMER0 Count Register

; Register Addresses Of TIMER1
M_TCSR1     EQU $FFFF8B             ; TIMER1 Control/Status Register
M_TLR1      EQU $FFFF8A             ; TIMER1 Load Reg
M_TCPR1     EQU $FFFF89             ; TIMER1 Compare Register
M_TCR1      EQU $FFFF88             ; TIMER1 Count Register

; Register Addresses Of TIMER2
M_TCSR2     EQU $FFFF87             ; TIMER2 Control/Status Register
M_TLR2      EQU $FFFF86             ; TIMER2 Load Reg
M_TCPR2     EQU $FFFF85             ; TIMER2 Compare Register
M_TCR2      EQU $FFFF84             ; TIMER2 Count Register
M_TPLR      EQU $FFFF83             ; TIMER Prescaler Load Register
M_TPCR      EQU $FFFF82             ; TIMER Prescalar Count Register

; Timer Control/Status Register Bit Flags
M_TE        EQU 0                   ; Timer Enable
M_TOIE      EQU 1                   ; Timer Overflow Interrupt Enable
M_TCIE      EQU 2                   ; Timer Compare Interrupt Enable
M_TC        EQU $F0                 ; Timer Control Mask (TC0-TC3)
M_INV       EQU 8                   ; Inverter Bit
M_TRM       EQU 9                   ; Timer Restart Mode
M_DIR       EQU 11                  ; Direction Bit
M_DI        EQU 12                  ; Data Input
M_DO        EQU 13                  ; Data Output
M_PCE       EQU 15                  ; Prescaled Clock Enable
M_TOF       EQU 20                  ; Timer Overflow Flag
M_TCF       EQU 21                  ; Timer Compare Flag

; Timer Prescaler Register Bit Flags
M_PS        EQU $600000             ; Prescaler Source Mask
M_PS0       EQU 21
M_PS1       EQU 22
```

```
; Timer Control Bits
M_TC0       EQU 4                   ; Timer Control 0
M_TC1       EQU 5                   ; Timer Control 1
M_TC2       EQU 6                   ; Timer Control 2
M_TC3       EQU 7                   ; Timer Control 3


;------------------------------------------------------------------------
;
;   EQUATES for Direct Memory Access (DMA)
;
;------------------------------------------------------------------------

; Register Addresses Of DMA
M_DSTR      EQU $FFFFF4             ; DMA Status Register
M_DOR0      EQU $FFFFF3             ; DMA Offset Register 0
M_DOR1      EQU $FFFFF2             ; DMA Offset Register 1
M_DOR2      EQU $FFFFF1             ; DMA Offset Register 2
M_DOR3      EQU $FFFFF0             ; DMA Offset Register 3

; Register Addresses Of DMA0
M_DSR0      EQU $FFFFEF             ; DMA0 Source Address Register
M_DDR0      EQU $FFFFEE             ; DMA0 Destination Address Register
M_DCO0      EQU $FFFFED             ; DMA0 Counter
M_DCR0      EQU $FFFFEC             ; DMA0 Control Register

; Register Addresses Of DMA1
M_DSR1      EQU $FFFFEB             ; DMA1 Source Address Register
M_DDR1      EQU $FFFFEA             ; DMA1 Destination Address Register
M_DCO1      EQU $FFFFE9             ; DMA1 Counter
M_DCR1      EQU $FFFFE8             ; DMA1 Control Register

; Register Addresses Of DMA2
M_DSR2      EQU $FFFFE7             ; DMA2 Source Address Register
M_DDR2      EQU $FFFFE6             ; DMA2 Destination Address Register
M_DCO2      EQU $FFFFE5             ; DMA2 Counter
M_DCR2      EQU $FFFFE4             ; DMA2 Control Register

; Register Addresses Of DMA4
M_DSR3      EQU $FFFFE3             ; DMA3 Source Address Register
M_DDR3      EQU $FFFFE2             ; DMA3 Destination Address Register
M_DCO3      EQU $FFFFE1             ; DMA3 Counter
M_DCR3      EQU $FFFFE0             ; DMA3 Control Register

; Register Addresses Of DMA4
M_DSR4      EQU $FFFFDF             ; DMA4 Source Address Register
M_DDR4      EQU $FFFFDE             ; DMA4 Destination Address Register
M_DCO4      EQU $FFFFDD             ; DMA4 Counter
M_DCR4      EQU $FFFFDC             ; DMA4 Control Register

; Register Addresses Of DMA5
M_DSR5      EQU $FFFFDB             ; DMA5 Source Address Register
M_DDR5      EQU $FFFFDA             ; DMA5 Destination Address Register
M_DCO5      EQU $FFFFD9             ; DMA5 Counter
M_DCR5      EQU $FFFFD8             ; DMA5 Control Register

; DMA Control Register
```

```
M_DSS       EQU $3              ; DMA Source Space Mask (DSS0-Dss1)
M_DSS0      EQU 0               ; DMA Source Memory space 0
M_DSS1      EQU 1               ; DMA Source Memory space 1
M_DDS       EQU $C              ; DMA Destination Space Mask (DDS-DDS1)
M_DDS0      EQU 2               ; DMA Destination Memory Space 0
M_DDS1      EQU 3               ; DMA Destination Memory Space 1
M_DAM       EQU $3f0            ; DMA Address Mode Mask (DAM5-DAM0)
M_DAM0      EQU 4               ; DMA Address Mode 0
M_DAM1      EQU 5               ; DMA Address Mode 1
M_DAM2      EQU 6               ; DMA Address Mode 2
M_DAM3      EQU 7               ; DMA Address Mode 3
M_DAM4      EQU 8               ; DMA Address Mode 4
M_DAM5      EQU 9               ; DMA Address Mode 5
M_D3D       EQU 10              ; DMA Three Dimensional Mode
M_DRS       EQU $F800           ; DMA Request Source Mask (DRS0-DRS4)
M_DCON      EQU 16              ; DMA Continuous Mode
M_DPR       EQU $60000          ; DMA Channel Priority
M_DPR0      EQU 17              ; DMA Channel Priority Level (low)
M_DPR1      EQU 18              ; DMA Channel Priority Level (high)
M_DTM       EQU $380000         ; DMA Transfer Mode Mask (DTM2-DTM0)
M_DTM0      EQU 19              ; DMA Transfer Mode 0
M_DTM1      EQU 20              ; DMA Transfer Mode 1
M_DTM2      EQU 21              ; DMA Transfer Mode 2
M_DIE       EQU 22              ; DMA Interrupt Enable bit
M_DE        EQU 23              ; DMA Channel Enable bit

; DMA Status Register
M_DTD       EQU $3F             ; Channel Transfer Done Status MASK (DTD0-DTD5)
M_DTD0      EQU 0               ; DMA Channel Transfer Done Status 0
M_DTD1      EQU 1               ; DMA Channel Transfer Done Status 1
M_DTD2      EQU 2               ; DMA Channel Transfer Done Status 2
M_DTD3      EQU 3               ; DMA Channel Transfer Done Status 3
M_DTD4      EQU 4               ; DMA Channel Transfer Done Status 4
M_DTD5      EQU 5               ; DMA Channel Transfer Done Status 5
M_DACT      EQU 8               ; DMA Active State
M_DCH       EQU $E00            ; DMA Active Channel Mask (DCH0-DCH2)
M_DCH0      EQU 9               ; DMA Active Channel 0
M_DCH1      EQU 10              ; DMA Active Channel 1
M_DCH2      EQU 11              ; DMA Active Channel 2


;------------------------------------------------------------------------
;
;   EQUATES for Phase Locked Loop (PLL)
;
;------------------------------------------------------------------------

; Register Addresses Of PLL
M_PCTL      EQU $FFFFFD         ; PLL Control Register

; PLL Control Register
M_MF        EQU $FFF            ; Multiplication Factor Bits Mask (MF0-MF11)
M_DF        EQU $7000           ; Division Factor Bits Mask (DF0-DF2)
M_XTLR      EQU 15              ; XTAL Range select bit
M_XTLD      EQU 16              ; XTAL Disable Bit
M_PSTP      EQU 17              ; STOP Processing State Bit
M_PEN       EQU 18              ; PLL Enable Bit
```

```
M_PCOD       EQU 19               ; PLL Clock Output Disable Bit
M_PD         EQU $F00000          ; PreDivider Factor Bits Mask (PD0-PD3)


;------------------------------------------------------------------------
;
;   EQUATES for BIU
;
;------------------------------------------------------------------------

; Register Addresses Of BIU
M_BCR        EQU $FFFFFB          ; Bus Control Register
M_DCR        EQU $FFFFFA          ; DRAM Control Register
M_AAR0       EQU $FFFFF9          ; Address Attribute Register 0
M_AAR1       EQU $FFFFF8          ; Address Attribute Register 1
M_AAR2       EQU $FFFFF7          ; Address Attribute Register 2
M_AAR3       EQU $FFFFF6          ; Address Attribute Register 3
M_IDR        EQU $FFFFF5          ; ID Register

; Bus Control Register
M_BA0W       EQU $1F              ; Area 0 Wait Control Mask (BA0W0-BA0W4)
M_BA1W       EQU $3E0             ; Area 1 Wait Control Mask (BA1W0-BA14)
M_BA2W       EQU $1C00            ; Area 2 Wait Control Mask (BA2W0-BA2W2)
M_BA3W       EQU $E000            ; Area 3 Wait Control Mask (BA3W0-BA3W3)
M_BDFW       EQU $1F0000          ; Default Area Wait Control Mask (BDFW0-BDFW4)
M_BBS        EQU 21               ; Bus State
M_BLH        EQU 22               ; Bus Lock Hold
M_BRH        EQU 23               ; Bus Request Hold

; DRAM Control Register
M_BCW        EQU $3               ; In Page Wait States Bits Mask (BCW0-BCW1)
M_BRW        EQU $C               ; Out Of Page Wait States Bits Mask (BRW0-BRW1)
M_BPS        EQU $300             ; DRAM Page Size Bits Mask (BPS0-BPS1)
M_BPLE       EQU 11               ; Page Logic Enable
M_BME        EQU 12               ; Mastership Enable
M_BRE        EQU 13               ; Refresh Enable
M_BSTR       EQU 14               ; Software Triggered Refresh
M_BRF        EQU $7F8000          ; Refresh Rate Bits Mask (BRF0-BRF7)
M_BRP        EQU 23               ; Refresh prescaler

; Address Attribute Registers
M_BAT        EQU $3               ; External Access Type and Pin Definition Bits
Mask(BAT0-BAT1)
M_BAAP       EQU 2                ; Address Attribute Pin Polarity
M_BPEN       EQU 3                ; Program Space Enable
M_BXEN       EQU 4                ; X Data Space Enable
M_BYEN       EQU 5                ; Y Data Space Enable
M_BAM        EQU 6                ; Address Muxing
M_BPAC       EQU 7                ; Packing Enable
M_BNC        EQU $F00             ; Number of Address Bits to Compare
Mask(BNC0-BNC3)
M_BAC        EQU $FFF000          ; Address to Compare Bits Mask (BAC0-BAC11)

; control and status bits in SR
M_CP         EQU $c00000          ; mask for CORE-DMA priority bits in SR
M_CA         EQU 0                ; Carry
M_V          EQU 1                ; Overflow
```

```
M_Z         EQU 2                   ; Zero
M_N         EQU 3                   ; Negative
M_U         EQU 4                   ; Unnormalized
M_E         EQU 5                   ; Extension
M_L         EQU 6                   ; Limit
M_S         EQU 7                   ; Scaling Bit
M_I0        EQU 8                   ; Interupt Mask Bit 0
M_I1        EQU 9                   ; Interupt Mask Bit 1
M_S0        EQU 10                  ; Scaling Mode Bit 0
M_S1        EQU 11                  ; Scaling Mode Bit 1
M_SC        EQU 13                  ; Sixteen_Bit Compatibility
M_DM        EQU 14                  ; Double Precision Multiply
M_LF        EQU 15                  ; DO-Loop Flag
M_FV        EQU 16                  ; DO-Forever Flag
M_SA        EQU 17                  ; Sixteen-Bit Arithmetic
M_CE        EQU 19                  ; Instruction Cache Enable
M_SM        EQU 20                  ; Arithmetic Saturation
M_RM        EQU 21                  ; Rounding Mode
M_CP0       EQU 22                  ; bit 0 of priority bits in SR
M_CP1       EQU 23                  ; bit 1 of priority bits in SR

; control and status bits in OMR
M_CDP       EQU $300                ; mask for CORE-DMA priority bits in OMR
M_MA        EQU 0                   ; Operating Mode A
M_MB        EQU 1                   ; Operating Mode B
M_MC        EQU 2                   ; Operating Mode C
M_MD        EQU 3                   ; Operating Mode D
M_EBD       EQU 4                   ; External Bus Disable bit in OMR
M_SD        EQU 6                   ; Stop Delay
M_MS        EQU 7                   ; Memory Switch bit in OMR
M_CDP0      EQU 8                   ; bit 0 of priority bits in OMR
M_CDP1      EQU 9                   ; bit 1 of priority bits in OMR
M_BEN       EQU 10                  ; Burst Enable
M_TAS       EQU 11                  ; TA Synchronize Select
M_BRT       EQU 12                  ; Bus Release Timing
M_ATE       EQU 15                  ; Address Tracing Enable bit in OMR.
M_XYS       EQU 16                  ; Stack Extension space select bit in OMR.
M_EUN       EQU 17                  ; Extensed stack UNderflow flag in OMR.
M_EOV       EQU 18                  ; Extended stack OVerflow flag in OMR.
M_WRP       EQU 19                  ; Extended WRaP flag in OMR.
M_SEN       EQU 20                  ; Stack Extension Enable bit in OMR.


;*********************************************************************************
;
;   EQUATES for 56302/3 interrupts
;
; Last update: June 11 1995
;
;*********************************************************************************
    page 132,55,0,0,0
    opt mex
intequ ident 1,0
    if @DEF(I_VEC)
                                    ;leave user definition as is.
```

```
    else
I_VEC       EQU $0
    endif

;-----------------------------------------------------------------------
; Non-Maskable interrupts
;-----------------------------------------------------------------------
I_RESET     EQU I_VEC+$00       ; Hardware RESET
I_STACK     EQU I_VEC+$02       ; Stack Error
I_ILL       EQU I_VEC+$04       ; Illegal Instruction
I_DBG       EQU I_VEC+$06       ; Debug Request
I_TRAP      EQU I_VEC+$08       ; Trap
I_NMI       EQU I_VEC+$0A       ; Non Maskable Interrupt

;-----------------------------------------------------------------------
; Interrupt Request Pins
;-----------------------------------------------------------------------
I_IRQA      EQU I_VEC+$10       ; IRQA
I_IRQB      EQU I_VEC+$12       ; IRQB
I_IRQC      EQU I_VEC+$14       ; IRQC
I_IRQD      EQU I_VEC+$16       ; IRQD

;-----------------------------------------------------------------------
; DMA Interrupts
;-----------------------------------------------------------------------
I_DMA0      EQU I_VEC+$18       ; DMA Channel 0
I_DMA1      EQU I_VEC+$1A       ; DMA Channel 1
I_DMA2      EQU I_VEC+$1C       ; DMA Channel 2
I_DMA3      EQU I_VEC+$1E       ; DMA Channel 3
I_DMA4      EQU I_VEC+$20       ; DMA Channel 4
I_DMA5      EQU I_VEC+$22       ; DMA Channel 5

;-----------------------------------------------------------------------
; Timer Interrupts
;-----------------------------------------------------------------------
I_TIM0C     EQU I_VEC+$24       ; TIMER 0 compare
I_TIM0OF    EQU I_VEC+$26       ; TIMER 0 overflow
I_TIM1C     EQU I_VEC+$28       ; TIMER 1 compare
I_TIM1OF    EQU I_VEC+$2A       ; TIMER 1 overflow
I_TIM2C     EQU I_VEC+$2C       ; TIMER 2 compare
I_TIM2OF    EQU I_VEC+$2E       ; TIMER 2 overflow

;-----------------------------------------------------------------------
; ESSI Interrupts
;-----------------------------------------------------------------------
I_SI0RD     EQU I_VEC+$30       ; ESSI0 Receive Data
I_SI0RDE    EQU I_VEC+$32       ; ESSI0 Receive Data With Exception Status
I_SI0RLS    EQU I_VEC+$34       ; ESSI0 Receive last slot
I_SI0TD     EQU I_VEC+$36       ; ESSI0 Transmit data
I_SI0TDE    EQU I_VEC+$38       ; ESSI0 Transmit Data With Exception Status
I_SI0TLS    EQU I_VEC+$3A       ; ESSI0 Transmit last slot
I_SI1RD     EQU I_VEC+$40       ; ESSI1 Receive Data
I_SI1RDE    EQU I_VEC+$42       ; ESSI1 Receive Data With Exception Status
I_SI1RLS    EQU I_VEC+$44       ; ESSI1 Receive last slot
I_SI1TD     EQU I_VEC+$46       ; ESSI1 Transmit data
I_SI1TDE    EQU I_VEC+$48       ; ESSI1 Transmit Data With Exception Status
I_SI1TLS    EQU I_VEC+$4A       ; ESSI1 Transmit last slot
```

```
;-----------------------------------------------------------------------
; SCI Interrupts
;-----------------------------------------------------------------------
I_SCIRD     EQU I_VEC+$50       ; SCI Receive Data
I_SCIRDE    EQU I_VEC+$52       ; SCI Receive Data With Exception Status
I_SCITD     EQU I_VEC+$54       ; SCI Transmit Data
I_SCIIL     EQU I_VEC+$56       ; SCI Idle Line
I_SCITM     EQU I_VEC+$58       ; SCI Timer

;-----------------------------------------------------------------------
; HOST Interrupts
;-----------------------------------------------------------------------
I_HRDF      EQU I_VEC+$60       ; Host Receive Data Full
I_HTDE      EQU I_VEC+$62       ; Host Transmit Data Empty
I_HC        EQU I_VEC+$64       ; Default Host Command

;-----------------------------------------------------------------------
; INTERRUPT ENDING ADDRESS
;-----------------------------------------------------------------------
I_INTEND    EQU I_VEC+$FF       ; last address of interrupt vector space
```

```
FALSE EQU 0
TRUE  EQU 1

```

```
; we place some convenient macros here to make our life easier:

PUSH_ACC        MACRO   ACC

        ;we're saving an accumulator to the stack.
        move    ACC\0,ssh       ;push acc0/acc1 onto stack in one location
        move    ACC\1,ssl       ;pushing acc0 already preincremented
        move    ACC\2,ssh       ; and then use another level for acc2

                ENDM

POP_ACC         MACRO   ACC

        ;we're retrieving an accumulator from the stack.
        move    ssh,ACC\2
        move    ssl,ACC\1      ;move acc1->acc1
        move    ssh,ACC\0      ;finish off with ACC1

                ENDM

START_CRITICAL  MACRO
        ;grab the interrupt bits and toss it on the stack
        movec   sr,ssh          ;toss it on the stack
        ori     #$03,mr         ;set the interrupt mask bits
                nop
                nop
                nop

        ;-- START OF CRITICAL CODE --
                ENDM

END_CRITICAL    MACRO
        ; -- END OF CRITICAL CODE --
        ;then reset the interrupt bits to normal
                nop
                nop
                nop

        movec   ssh,sr          ;back to normal
                ENDM

FORCE_INTERRUPTS_ON MACRO
        ;we force interrupts on
        andi    #$fc,mr         ;set SR mask bits to 0, unmasking all interrupts
                ENDM

FORCE_INTERRUPTS_OFF MACRO
        ;we force interrupts off
        ori     #$03,mr         ;set SR mask bits to 3, masking all interrupts
                ENDM

```

```
###############################################################################
#
# Makefile
#
# EE/CS 52: DSP56K MP3/Ogg Jukebox Project
#
# Revision History
# 2/16/2010  Glen George     Initial Revision
# 3/10/2011  Raymond Jimenez now working for our build
#
################################################################################


### VARIABLES ##################################################################

# C Compiler Definition
CC=g563c
ASM=asm56300
LINK=dsplnk
HEXMAKE=srec

# Directory Definitions
SRCDIR=Z:\src
OBJDIR=Z:\obj
SYSDIR=$(SRCDIR)
JUKEDIR=$(SRCDIR)
TOOLDIR=c:\DSP56K\BIN

# key files
CRT0=crt0.cln

# Compiler Flags
CFLAGS= -I$(SYSDIR) -IC:\DSP56K\include -BC:\DSP56K\lib -DDSP56K -mx-memory \
-Wall -alo -O
ASMFLAGS=


# Jukebox files

JUKEFILES=$(wildcard $(JUKEDIR)/*.c)
JUKEOBJS=$(notdir $(JUKEFILES:.c=.cln))


# Linker Command File

LINKCMD=jukebox.cli


# low-level assembly files

# user must define the symbol SYSOBJS below
# example definition:
# SYSOBJS= keypad.cln display.cln
SYSOBJS= stubfncs.cln intrrpts.cln queues.cln keyfunc.cln \
 converts.cln timing.cln ide.cln display.cln sound.cln
```

```
### RULES ##############################################################

# Specify Virtual Path for Object Files
vpath %.cln $(OBJDIR)

# Rule for Building Objects from C Sources
%.cln:
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.c,$(<)) -o $(OBJDIR)\$@



### COMMAND LINE TARGETS ###############################################

# Target for Building All Code
#juke: $(SYSOBJS) $(JUKEOBJS)
#	$(LINK) -f$(LINKCMD)
#	$(HEXMAKE) -b jukebox.cld

jukebox.cld: $(SYSOBJS) $(JUKEOBJS)
	$(CC) $(CFLAGS) -crt $(OBJDIR)\$(CRT0) -o $(OBJDIR)\jukebox.cld -j -M $(^)
	srec -b $(OBJDIR)\jukebox.cld

# Target for Cleaning House
.PHONY: clean
clean:
	del $(OBJDIR)\jukebox.cld $(OBJDIR)\*.cln



### GENERAL TARGETS ####################################################

# Targets for Jukebox Code
mainloop.cln : $(SYSDIR)/mainloop.c $(SYSDIR)/interfac.h $(SYSDIR)/mp3defs.h \
                $(SYSDIR)/keyproc.h $(SYSDIR)/updatfnc.h \
                $(SYSDIR)/trakutil.h $(SYSDIR)/fatutil.h

playmp3.cln  : $(SYSDIR)/playmp3.c $(SYSDIR)/mp3defs.h $(SYSDIR)/keyproc.h \
                $(SYSDIR)/updatfnc.h $(SYSDIR)/trakutil.h $(SYSDIR)/fatutil.h

ffrev.cln    : $(SYSDIR)/ffrev.c $(SYSDIR)/mp3defs.h $(SYSDIR)/keyproc.h \
                $(SYSDIR)/updatfnc.h $(SYSDIR)/trakutil.h $(SYSDIR)/fatutil.h

keyupdat.cln : $(SYSDIR)/keyupdat.c $(SYSDIR)/interfac.h $(SYSDIR)/mp3defs.h \
                $(SYSDIR)/keyproc.h $(SYSDIR)/updatfnc.h \
                $(SYSDIR)/trakutil.h $(SYSDIR)/fatutil.h

trakutil.cln : $(SYSDIR)/trakutil.c $(SYSDIR)/interfac.h $(SYSDIR)/mp3defs.h \
                $(SYSDIR)/trakutil.h $(SYSDIR)/fatutil.h $(SYSDIR)/vfat.h

fatutil.cln  : $(SYSDIR)/fatutil.c $(SYSDIR)/mp3defs.h $(SYSDIR)/interfac.h \
                $(SYSDIR)/vfat.h $(SYSDIR)/fatutil.h

stubfncs.cln : $(SYSDIR)/stubfncs.c $(SYSDIR)/mp3defs.h

simide.cln   : $(SYSDIR)/simide.c $(SYSDIR)/mp3defs.h $(SYSDIR)/interfac.h
```

```
test_dram.cln: $(SYSDIR)/test_dram.c

# Targets for Low-Level Code
# user must supply these targets and dependencies
# example defintions:
#    keypad.cln: $(SYSDIR)/keypad.asm $(SYSDIR)/general.inc $(SYSDIR)/keypad.inc
#            $(ASM) $(ASMFLAGS) -b$(OBJDIR)/keypad.cln $(SYSDIR)/keypad.asm

crt0.cln : $(SYSDIR)/crt0.asm
	$(CC) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

intrrpts.cln  : $(SYSDIR)/intrrpts.asm $(SYSDIR)/display.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

display.cln  : $(SYSDIR)/display.asm $(SYSDIR)/display.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

queues.cln  : $(SYSDIR)/queues.asm
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

keyfunc.cln  : $(SYSDIR)/keyfunc.asm $(SYSDIR)/keyfunc.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

converts.cln  : $(SYSDIR)/converts.asm $(SYSDIR)/converts.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

timing.cln  : $(SYSDIR)/timing.asm $(SYSDIR)/timing.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

ide.cln  : $(SYSDIR)/ide.asm $(SYSDIR)/ide.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@

sound.cln  : $(SYSDIR)/sound.asm $(SYSDIR)/sound.inc
	$(CC) $(CFLAGS) -c $(patsubst %.cln,%.asm,$(<)) -o $(OBJDIR)\$@
```

Chapter 9

Release Notes/Errata

9.1 Known Bugs

- Due to bad soldering, the DRAM may not be always functional. To resolve this issue, one can resort to percussive maintenance or a "hot-plug" of the DRAM.

  In order to hotplug the DRAM, start the system as described in the Quick-Start guide. Begin playing a file—you may notice that the file skips more than usual. At some poing during playback, carefully push apart on the two metal latches holding the DRAM in.

  Carefully angle the DRAM out of its socket, and remove it, taking care to prevent shorting out adjacent pins. Once this is done, carefully place the DRAM back in its slot and tilt it back until the latches lock in place.

  The cause of this issue is believed to be solder quality issues/joint issues; wiring and contacts are not the problem. No fix is known at this time.

- In rare circumstances, interrupts can exchange order due to DSP scheduling issues, resulting in a corrupted looking display. When this happens, a song title may be displayed with an incorrect artist, or the same with titles.

  In order to correct this, continue browsing through songs in order to refresh the display; the display should restore to its proper state after cycling through a couple songs.

No fix is known at this time.

- Adding songs during Daylight Saving Time may not operate correctly. You may see songs with extremely long playtimes, such as 500 minutes. This is due to a bug in the MS-DOS song-adding program, which does not operate correctly given Daylight Saving Time.

  In order to work around this, one can modify the file's playtime data directly by using `touch` from a GNU utilities collection.[1]

  Simply touch the file, after running `makesong` on it, with a modification time equivalent to the length of the song after midnight.

  In other words, if the song is 4 minutes and 30 seconds long, simply change the modification time on the song to 04:30AM, on any day.

  No fix is known at this time.

9.2 Known Limitations

- Due to the necessity to byte-reformat the IDE data, the system's bandwith is somewhat reduced. Playback of WAV files is not currently supported, and will result in stuttered playback.

  This issue is not believed to affect formats with comparable bitrates to MP3, such as Ogg Vorbis, WMA, AAC, or FLAC.

  A future software release may remove this limitation.

[1] `http://unxutils.sourceforge.net/`