

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 output 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

Introduction	iii
I Quick Start Guide	1
1 Quick Start Guide	3
II User Guide	7
2 Basic Usage	9
2.1 The Display	9
2.2 Playing Songs	10
2.3 Volume Control	10
3 Advanced Features	11
3.1 Adding Songs	11
3.2 Playing non-MP3 files	12
III Developer Documentation	13
4 System Overview	15
4.1 Introduction to Architecture	15
4.2 MP3 data flow	15
4.3 MP3 Decoder	16
4.4 DAC/Analog Stage	18
4.4.1 Power	18
4.4.2 DAC	18

4.5	IDE	19
4.6	DRAM	19
5	Memory Map and Registers	23
6	Timing	27
6.1	Timing Targets	27
6.1.1	SRAM	27
6.1.2	ROM	34
6.1.3	DRAM	38
6.1.4	IDE	45
6.1.5	DAC (via SCI)	49
6.1.6	MP3 Decoder (via SCI)	53
6.1.7	LCD Display	57
6.2	Timing Simulation (CPLD)	57
7	Schematics	59
7.1	Prototyping Board/CPU	59
7.2	CPLD	61
7.3	MP3 Decoder Daughterboard	63
7.4	DAC/Analog Out	65
7.5	Display/Keypad	67
7.6	IDE Interface	69
7.7	DRAM & Address Multiplexing	71
8	Annotated Code	73
8.1	CPLD Code	73
8.2	Bootup Code (crt0.asm)	87
8.3	Queues (queues.asm)	95
8.4	Display (display.inc, display.asm)	105
8.5	Keypad (keyfunc.inc, keyfunc.asm)	124
8.6	IDE Interface	133
8.7	Sound (DAC & MP3 Decoder)	141
8.8	Timing	186
8.9	Interrupts	192
8.10	User Interface	202
8.11	Miscellaneous	222

9	Release Notes/Errata	241
9.1	Known Bugs	241
9.2	Known Limitations	242

List of Figures

1.1	Main MP3 Playback Unit	4
1.2	MP3 Player's Keypad	5
1.3	LCD Display Unit	6
2.1	Typical Appearance of the LCD Display	9
4.1	Overall System Block Diagram	16
4.2	Main Board Components	17
4.3	DRAM Block Diagram	20

List of Tables

4.1	DAC Startup Mode	18
-----	----------------------------	----

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!

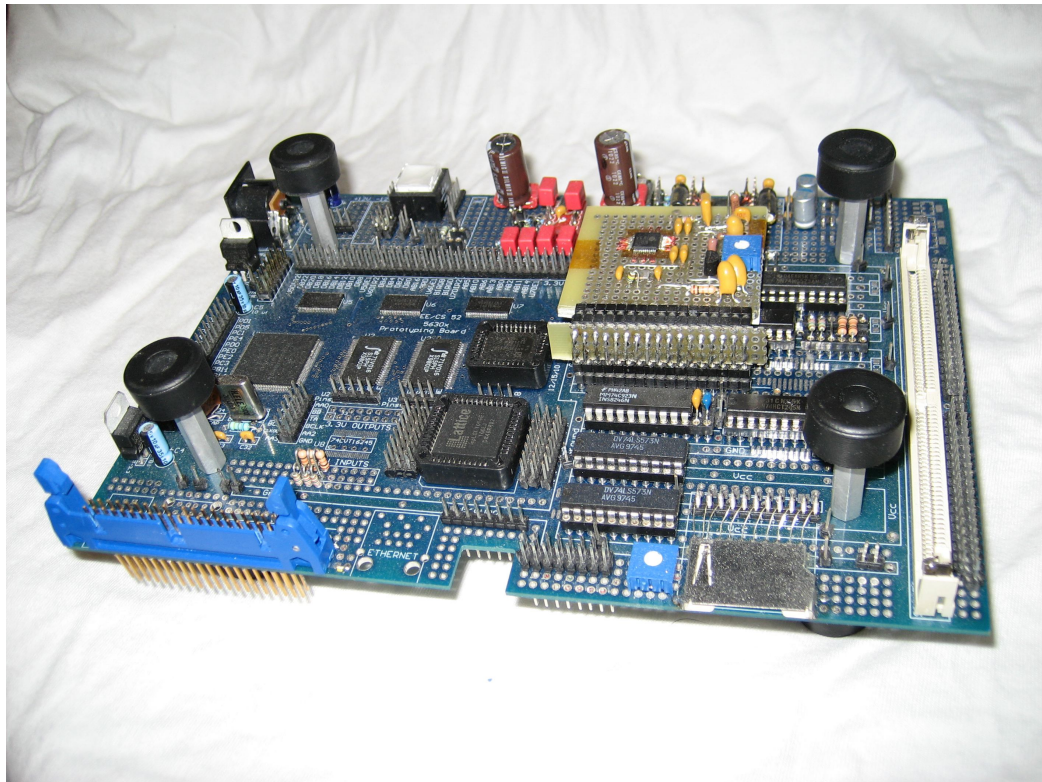


Figure 1.1: Main MP3 Playback Unit

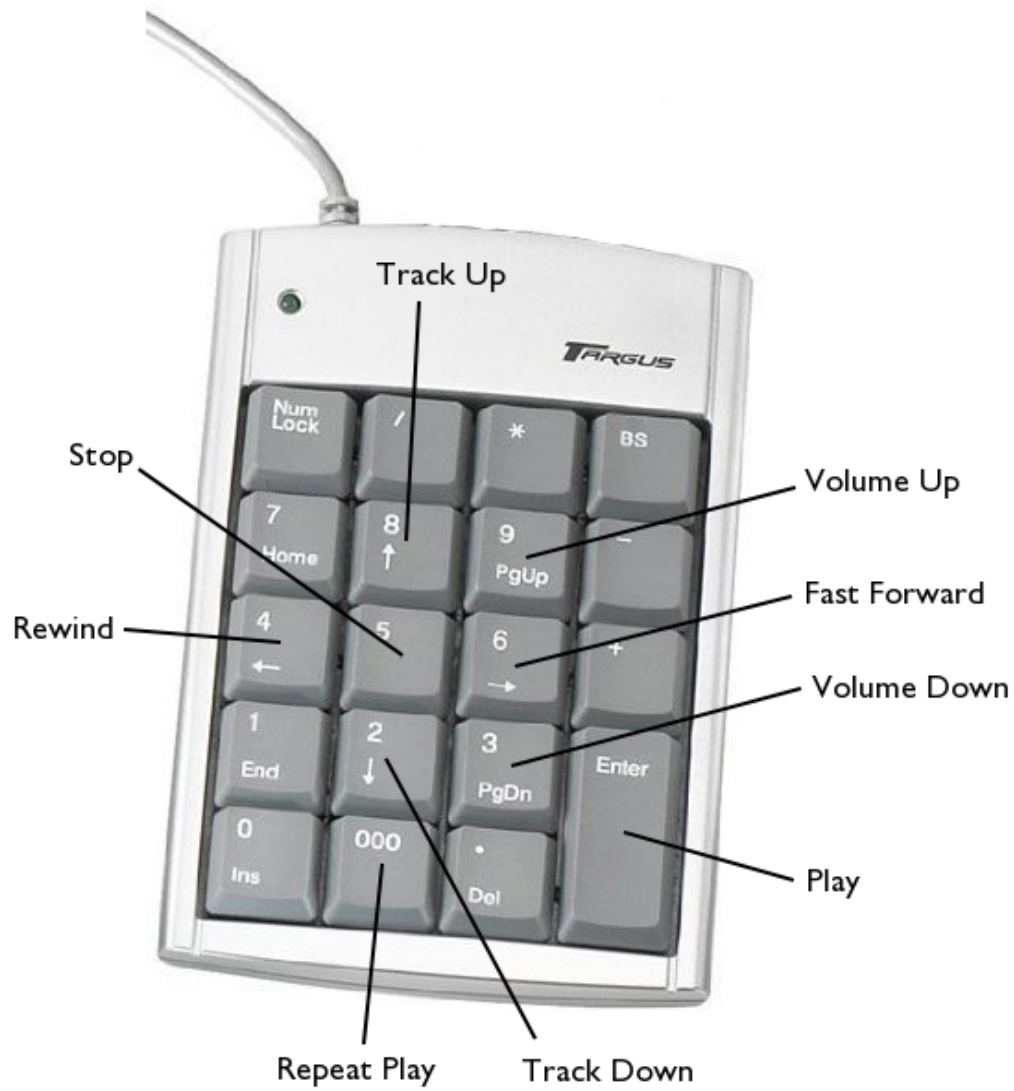


Figure 1.2: MP3 Player's Keypad

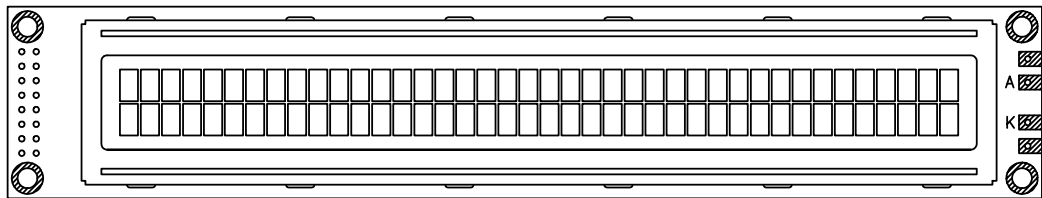


Figure 1.3: LCD Display Unit

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.

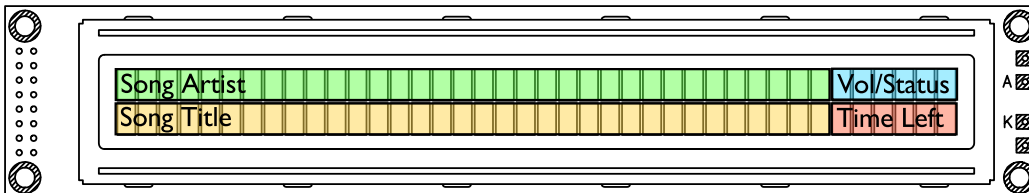


Figure 2.1: Typical Appearance of the LCD Display

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.

¹If 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).

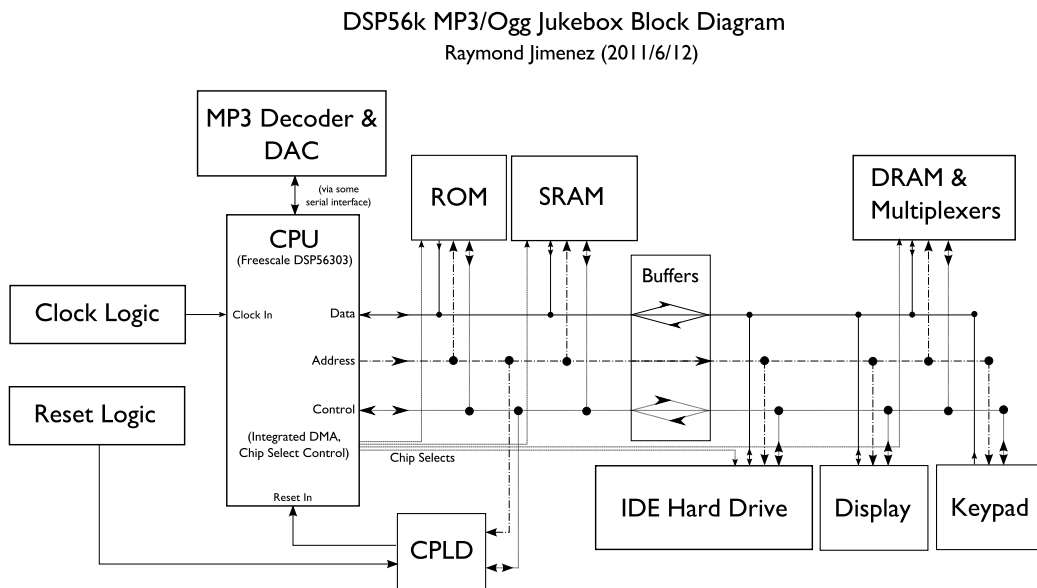


Figure 4.1: Overall System Block Diagram

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 transferred to the decoder, it decompresses the file and outputs the audio data via I²S 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

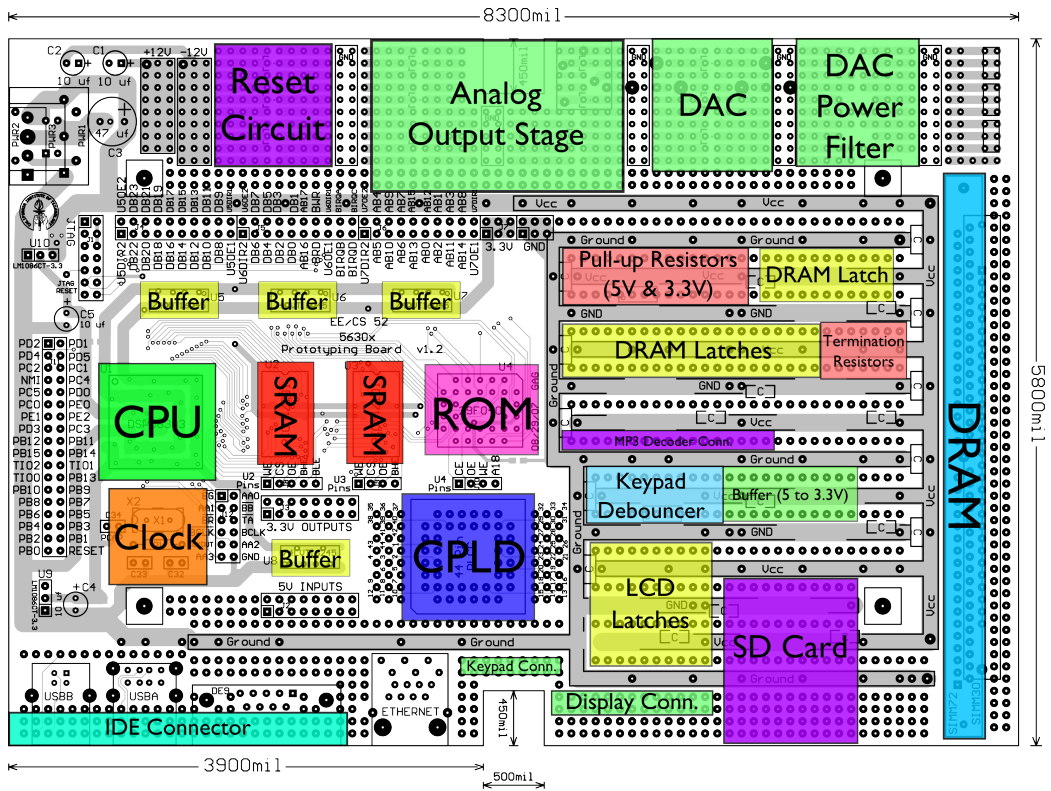


Figure 4.2: Main Board Components

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 Laboratories¹ 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 $1\mu\text{F}$ of capacitance 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 I²S output is 96kHz, 16-bit stereo with a 12.288MHz master clock ($\text{MCLK} = 128fs$), 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

DRAM Block Diagram

Raymond Jimenez (2011/06/12)

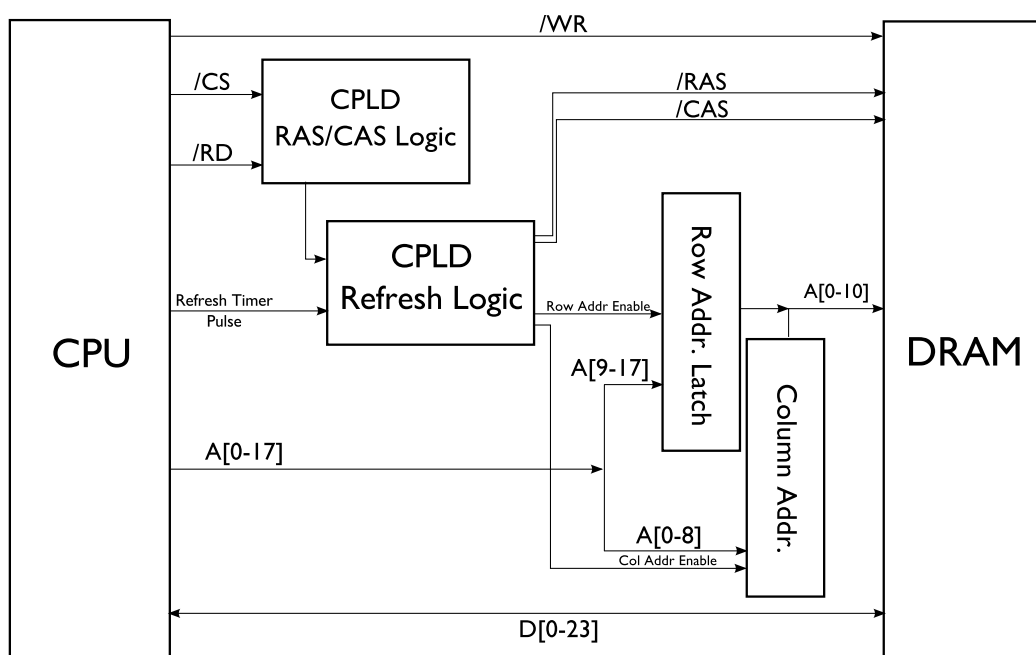


Figure 4.3: DRAM Block Diagram

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

[illegible]

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	

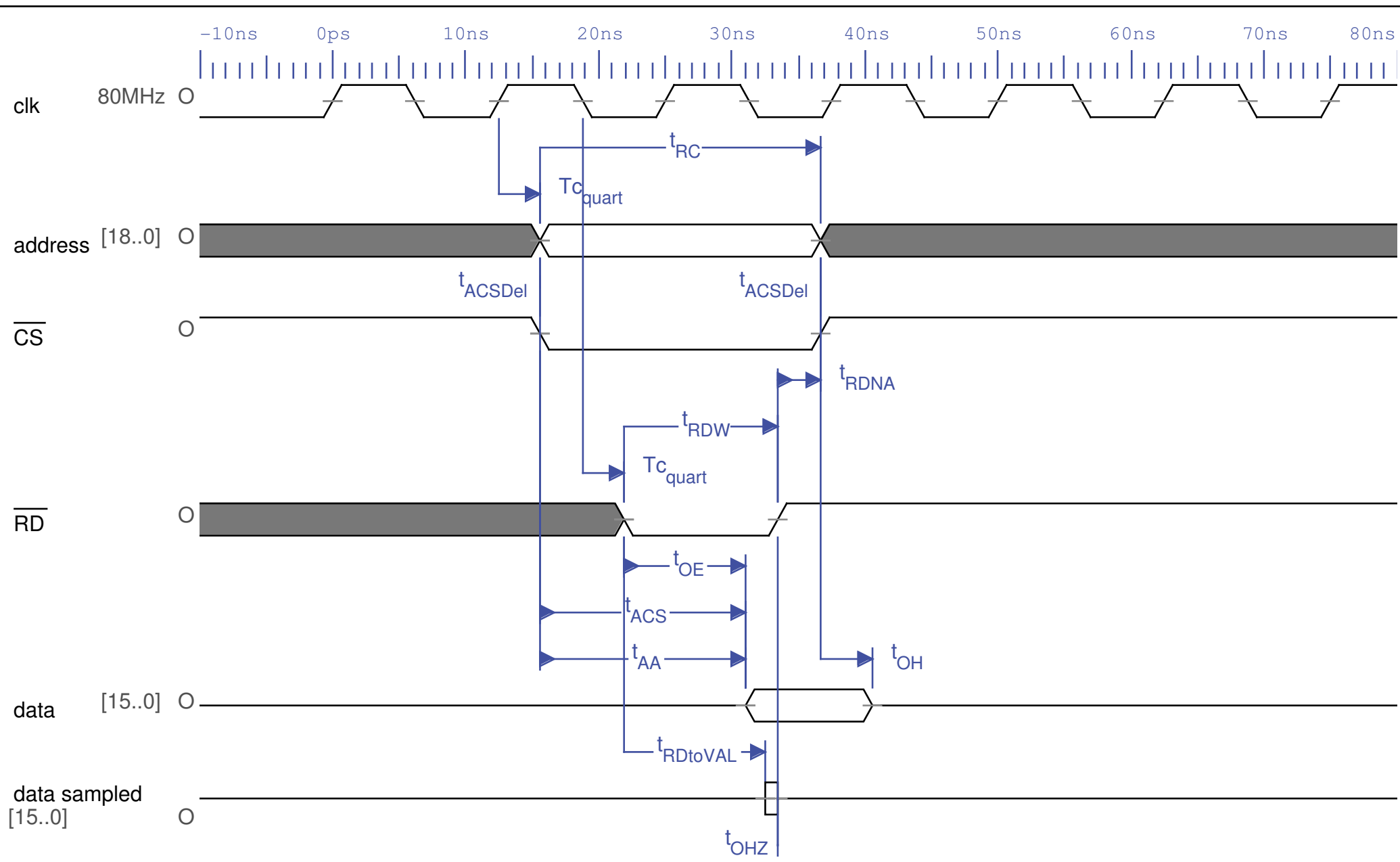
Chapter 6

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

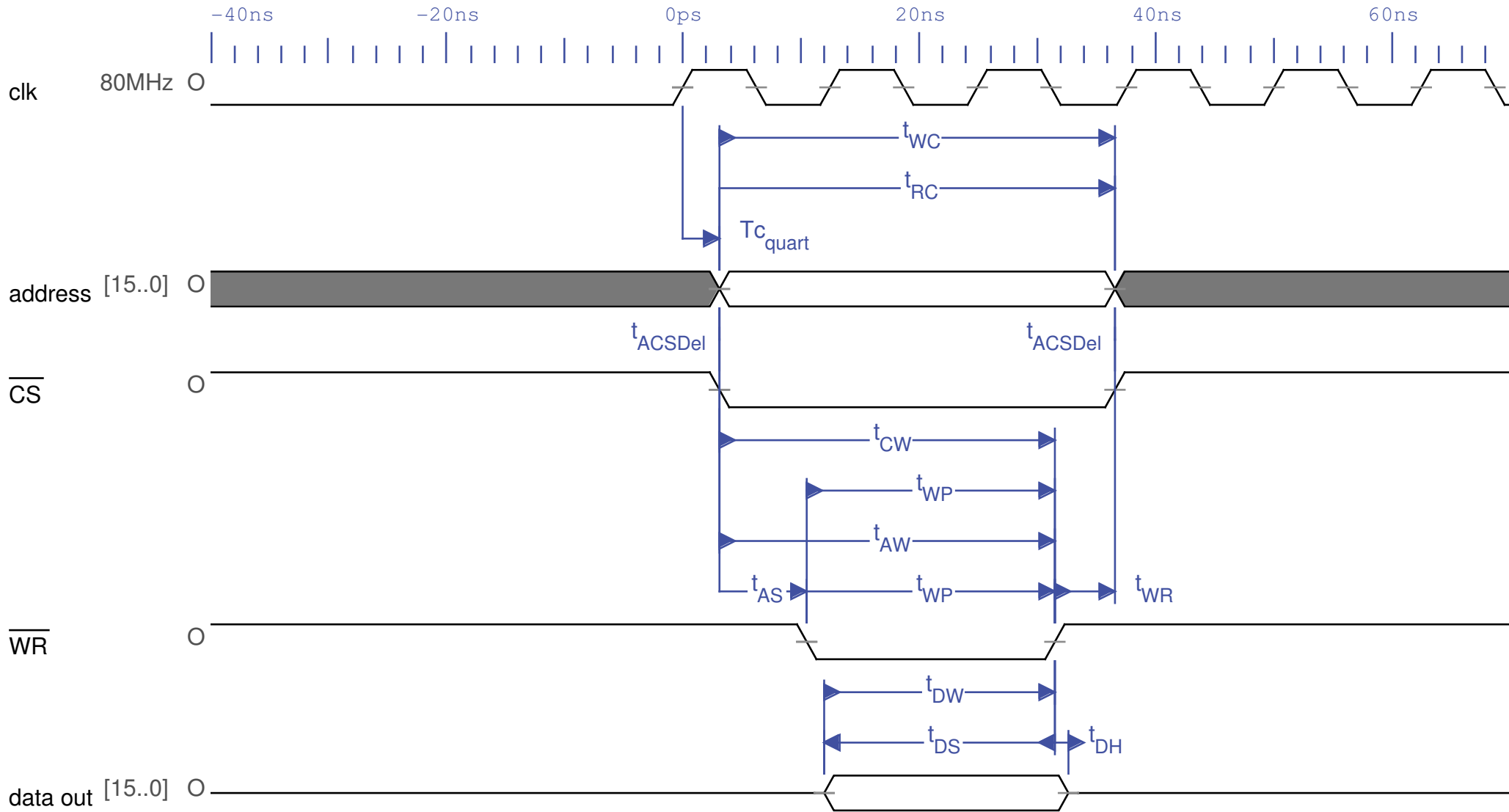


SRAM Read, CPU

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
T _c		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
T _{c_{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}		$\overline{\text{RD}}$ 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 $\overline{\text{RD}}$ to address valid	-2.875ns	-2.875ns	-2.875ns
t _{RDDE}	=ENG((0 * NOM("Tc")) + (3 * NOM("Tc") / 4) - 4000)	$\overline{\text{RD}}$ deassertion time	5.375ns		5.375ns
t _{RDNA}	=ENG((NOM("Tc") / 4 - 2000))	$\overline{\text{RD}}$ deassertion to address not valid	1.125ns		1.125ns
t _{RDW}	=ENG(((NOM("Ws") * NOM("Tc")) + (NOM("Ws") * (NOM("Tc") / 4)) - 4000))	$\overline{\text{RD}}$ pulse width	11.625ns		11.625ns
t _{RDtoVAL}	=ENG((NOM("Tc") * NOM("Ws")) + (NOM("Ws") * NOM("Tc") / 4) - 5000)	$\overline{\text{RD}}$ 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		



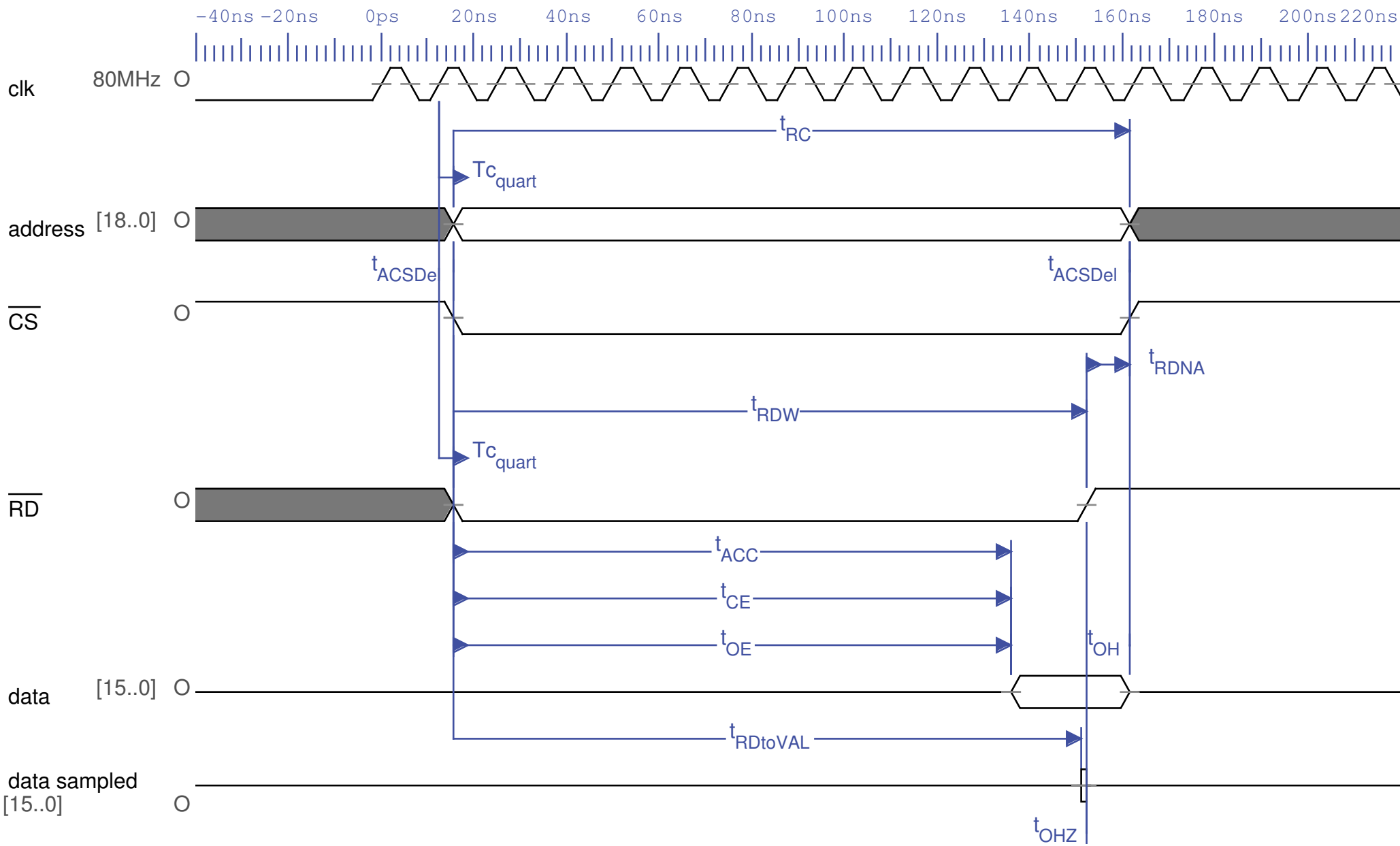
SRAM Write, CPU

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
T _c		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
T _{c_{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)-2 000)	$\overline{WR}$ 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)-200 0)	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

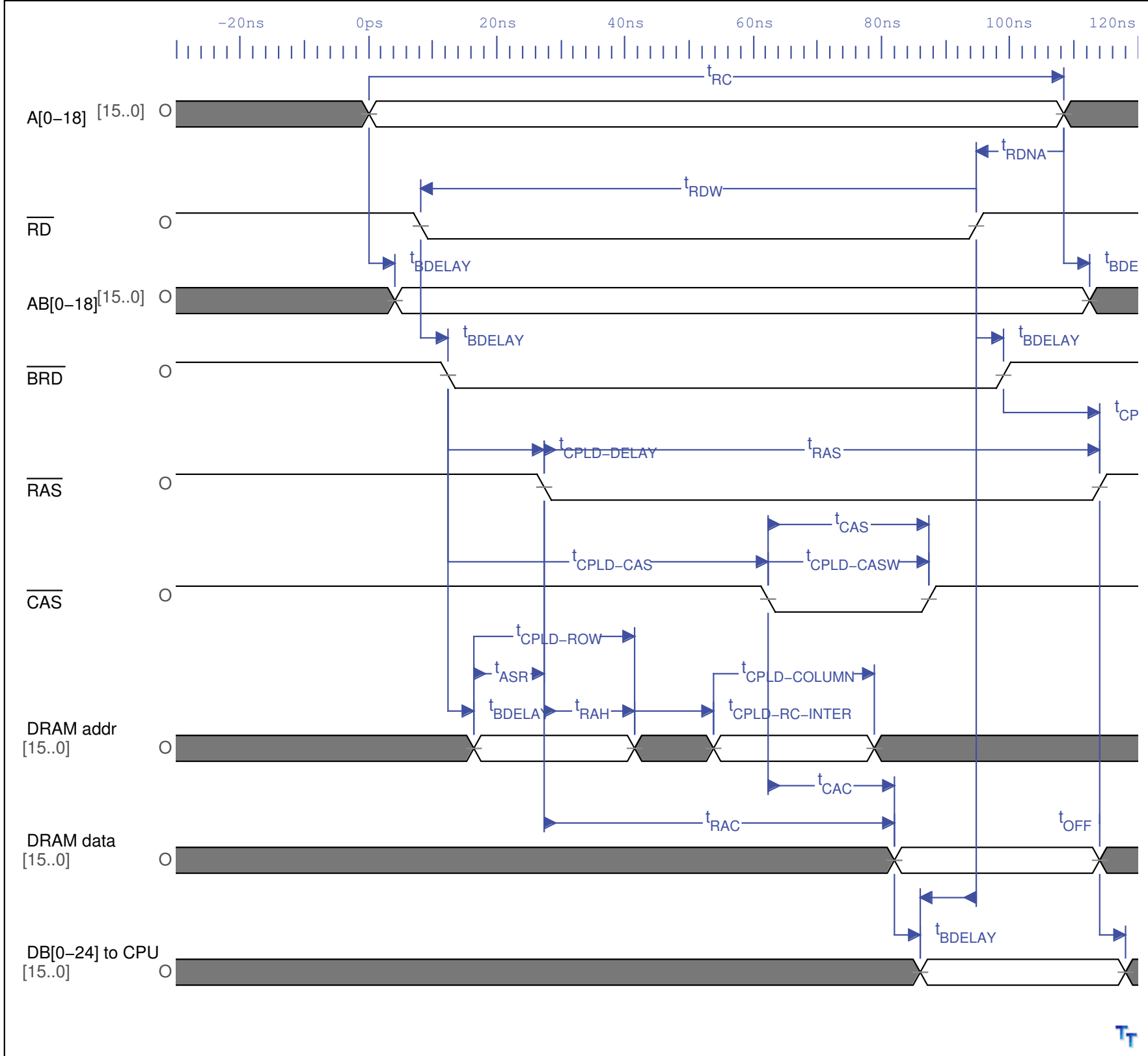


ROM Read, CPU					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
T _c		Clock Period (80MHz)	12.5ns	12.5ns	12.5ns
T _{c_{quant}}	=ENG(NOM("Tc")/4)	1/4 clock period	3.125ns	3.125ns	3.125ns
Ws		wait states			9
t _{ACSDel}		address to chip select delay	0ps	0ps	0
t _{OHZ}		$\overline{\text{RD}}$ deassertion to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

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		

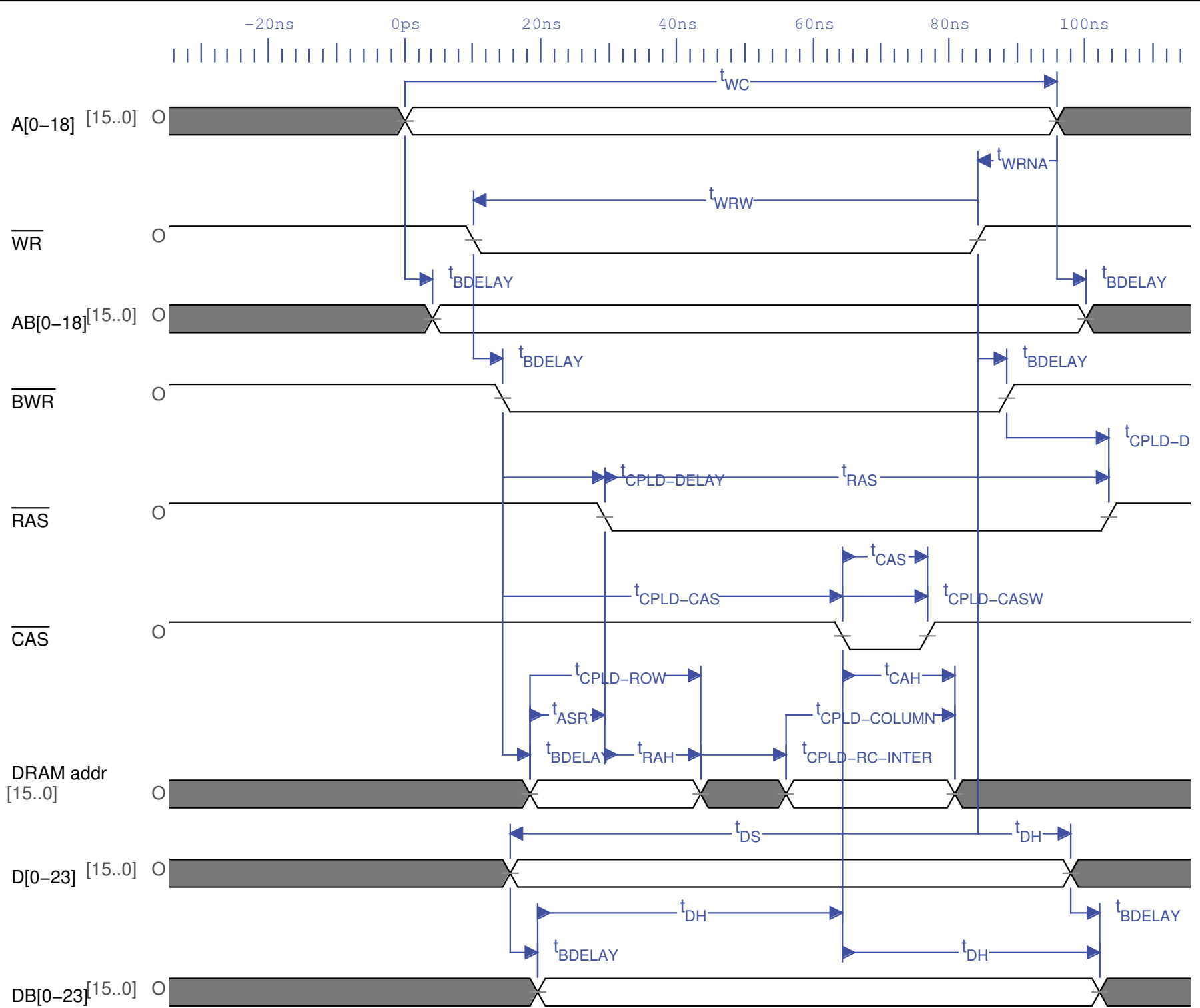
6.1.3 DRAM



CPU timing					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{BDELAY}		general buffer delay	4.1ns		
t_{RC}		address cycle time	108.5ns		
T_c		clock period	12.5ns		
t_{RDNA}		117 - RD deassertion to address not valid	-13.625ns		
W_s		wait states	7		
$t_{CPLD-DELAY}$		average CPLD gate delay for combinatorials	15ns		
$t_{CPLD-RC-INTER}$		intervening time between row	12.5ns		
Inter cycles		#CPU cycles between row and column addr	1		
$t_{CPLD-CAS}$		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		
$t_{CPLD-CASW}$		CAS length (CPLD synth)	25ns		
CAS length cycles		#CPU cycles for CAS length	2		
$t_{CPLD-ROW}$		row address length triggered from	25ns		
$t_{CPLD-COLUMN}$		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	

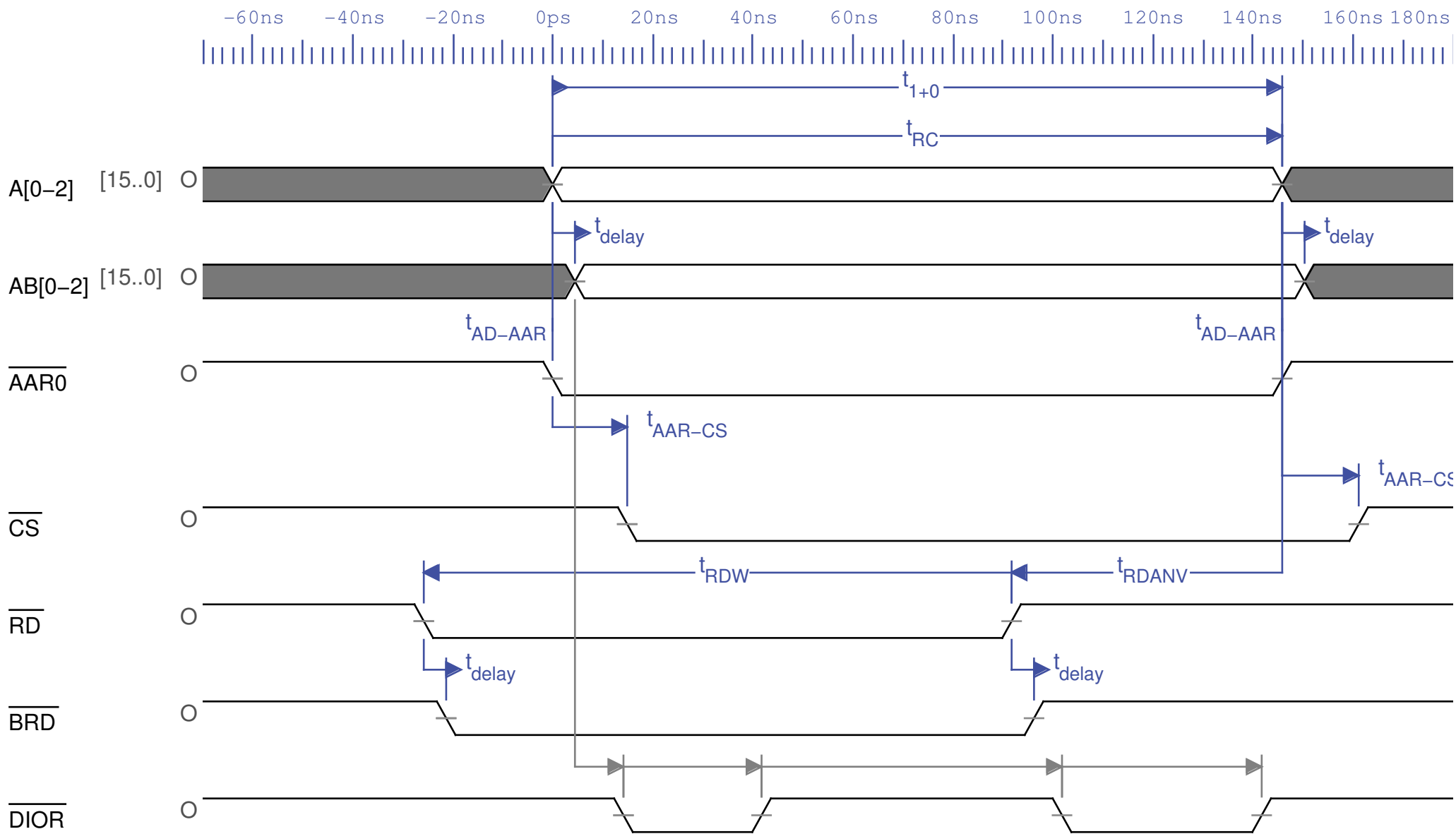


CPU timing					
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{BDELAY}		general buffer delay	4.1ns		
t_{DS}		data setup time	-68.875ns		
t_{WC}		address cycle time	96ns		
t_{DH}		data hold time	13.625ns		
T_c		clock period	12.5ns		
t_{WRNA}		103 - WR deassertion to address not valid	-11.625ns		
W_s		wait states	6		
$t_{CPLD-DELAY}$		average CPLD gate delay for combinatorials	15ns		
$t_{CPLD-RC-INTER}$		intervening time between row	12.5ns		
Inter cycles		#CPU cycles between row and column addr	1		
$t_{CPLD-CAS}$		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		
$t_{CPLD-CASW}$		CAS length (CPLD synth)	12.5ns		
CAS length cycles		#CPU cycles for CAS length	1		
$t_{CPLD-ROW}$		row address length triggered from	25ns		
$t_{CPLD-COLUMN}$		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



CPU timings

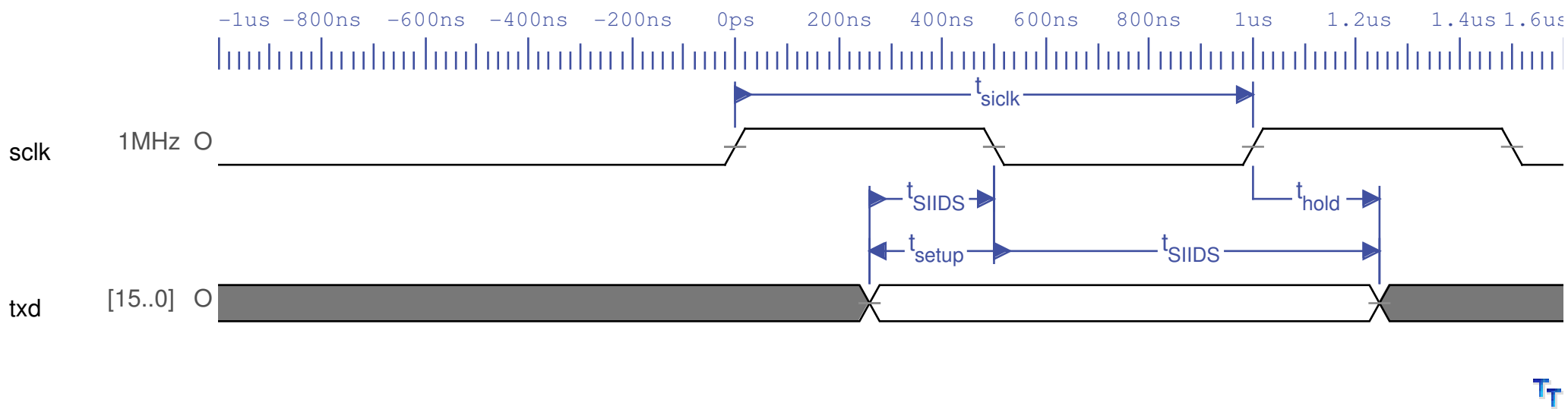
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{delay}		address buffer	4.5ns		
$t_{\text{AD-AAR}}$		delay between address bus and AAR	0	0	
$t_{\text{AAR-CS}}$		delay between AAR and CS (CPLD)	15ns		
T_c		CPU clock period		12.5ns	
W_s		Wait states	9		
t_{RDANV}		RD deassertion to address not valid	-54.25ns		
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 parameters to fit DAC timing.



CPU output

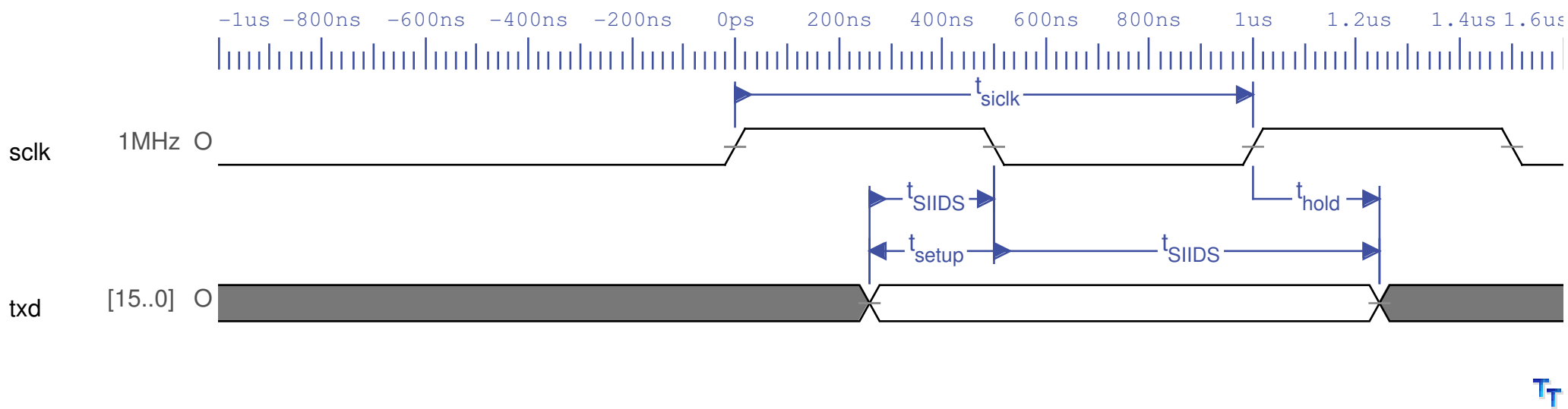
SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
T _c		internal clock	12.5ns		12.5ns
t _{hold}		hold time after rising edge	243.75ns		
t _{sc}		serial clock period	1us		1us
t _{setup}		setup time before falling edge	-239.25ns		

decoder requirements

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
t_{SCLK}		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 parameters to fit MP3 decoder timing. Additionally, SCI did not support DMA, so we switched to ESSI for a speed boost.



CPU output

SYMBOL	DEFINITION	DESCRIPTION	MIN	MAX	NOM
T _c		internal clock	12.5ns		12.5ns
t _{hold}		hold time after rising edge	243.75ns		
t _{sc}		serial clock period	1us		1us
t _{setup}		setup time before falling edge	-239.25ns		

decoder requirements

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

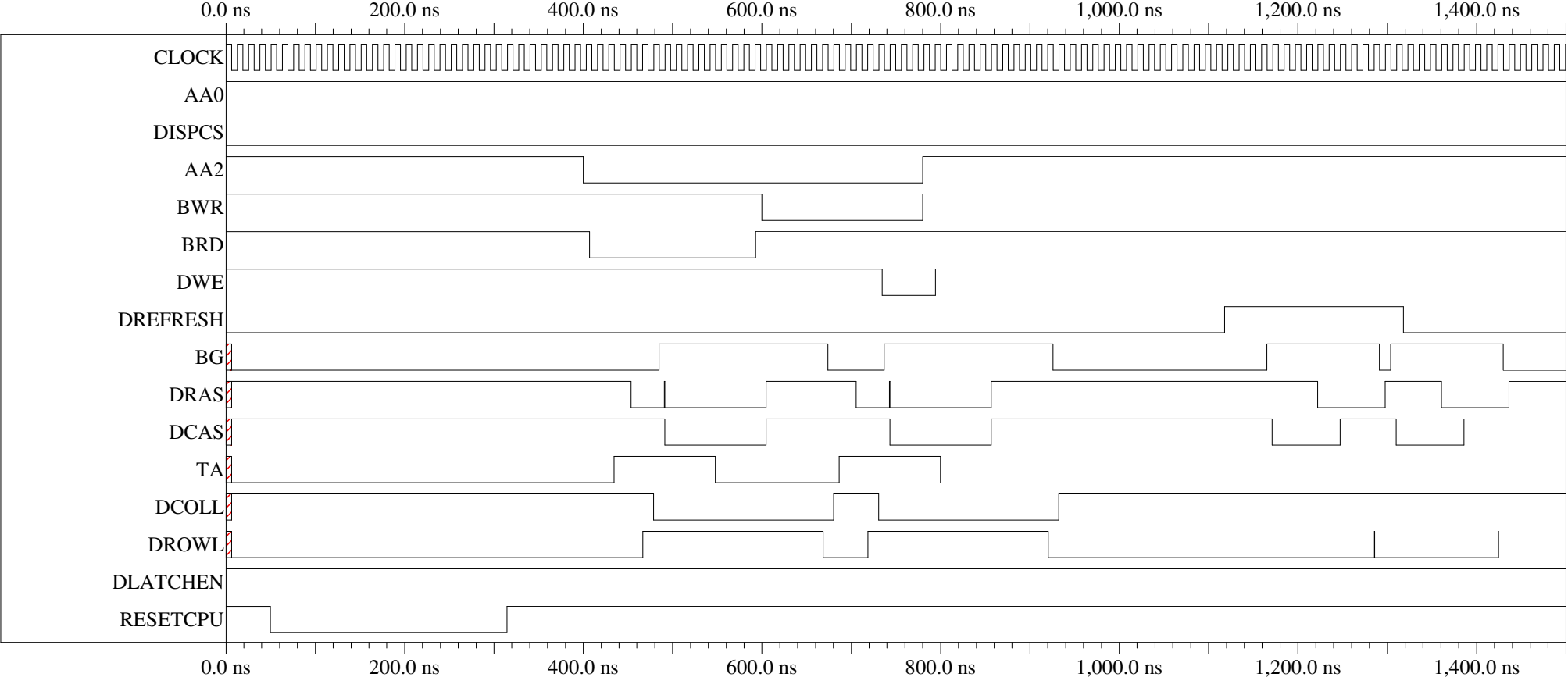
6.1.7 LCD Display

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:

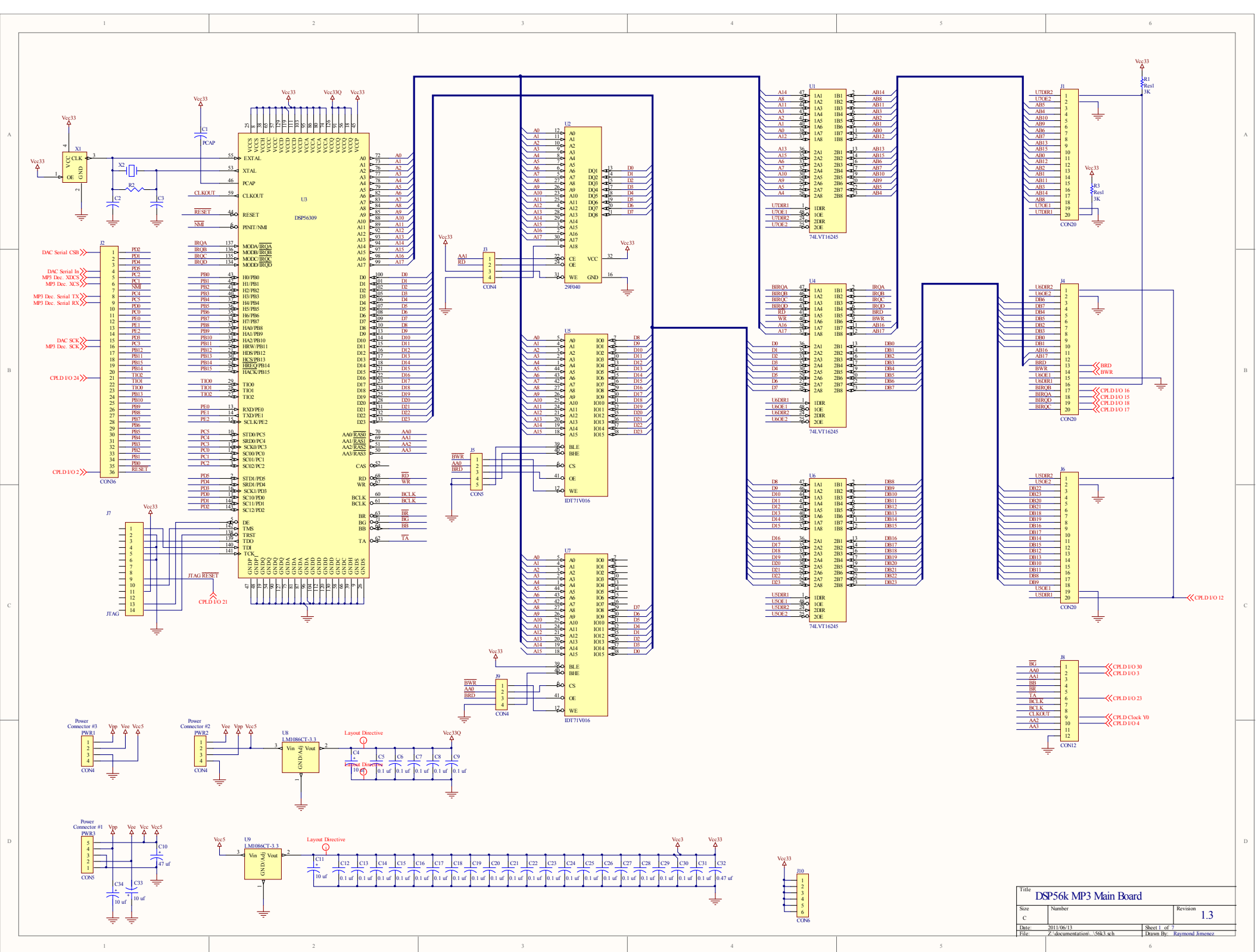
- a reset event
- a DRAM read followed immediately by a DRAM write
- two DRAM refresh cycles



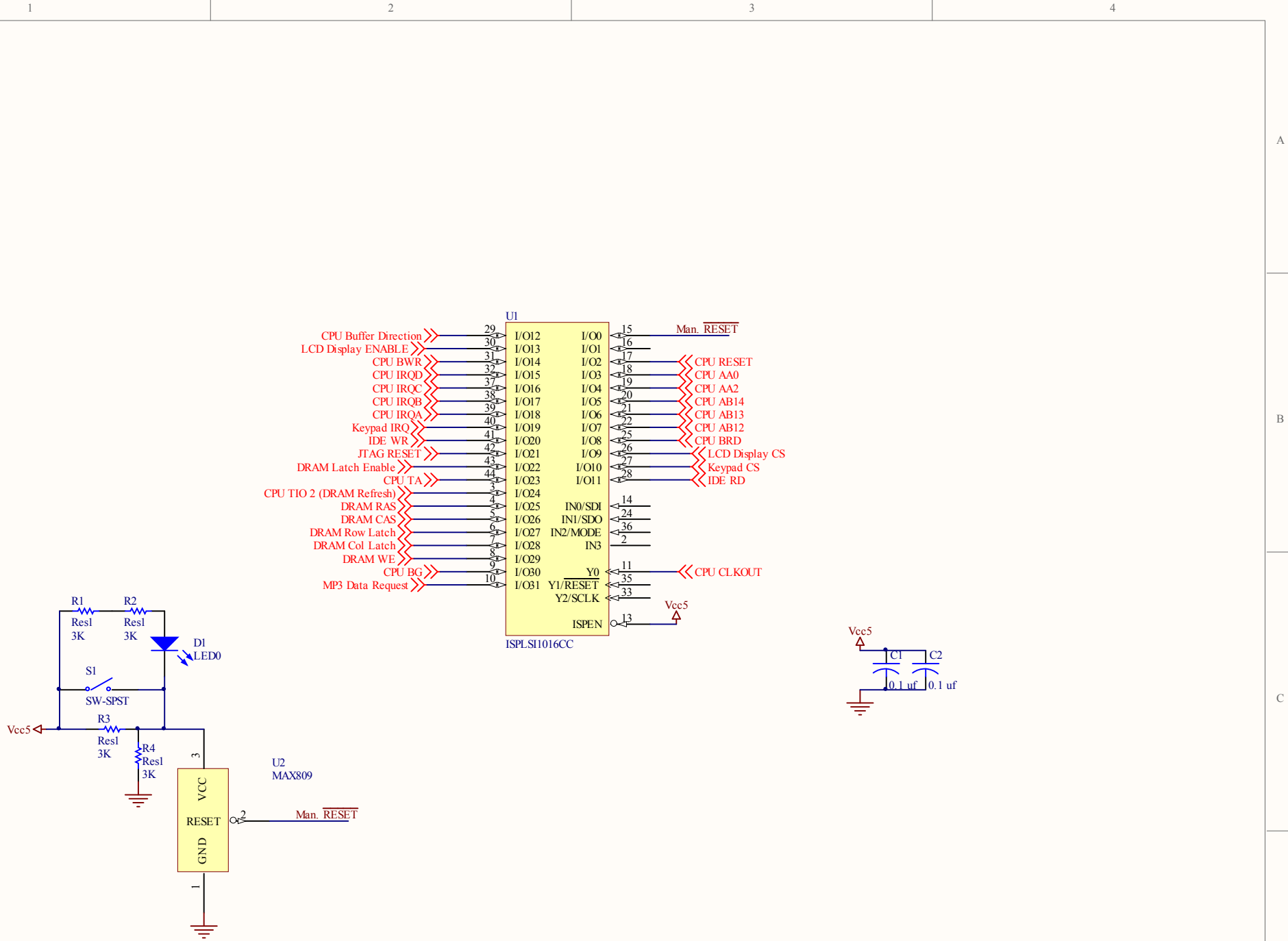
Chapter 7

Schematics

7.1 Prototyping Board/CPU

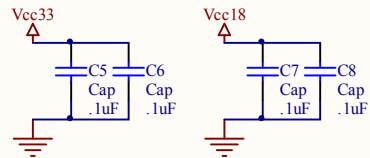
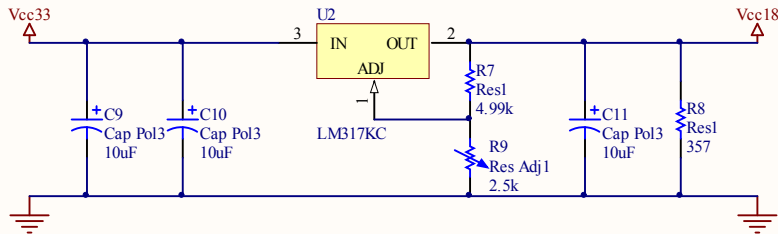
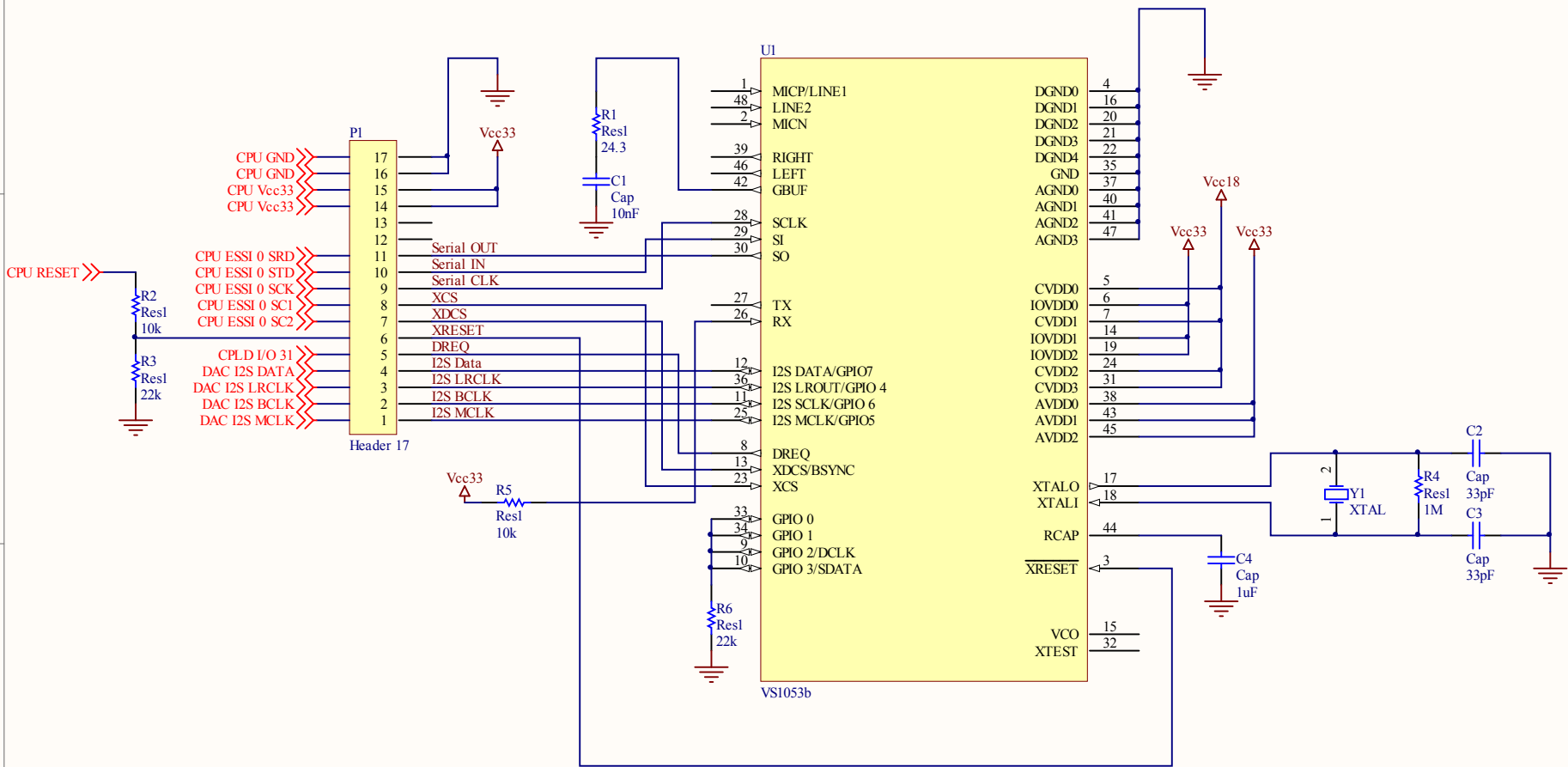


7.2 CPLD



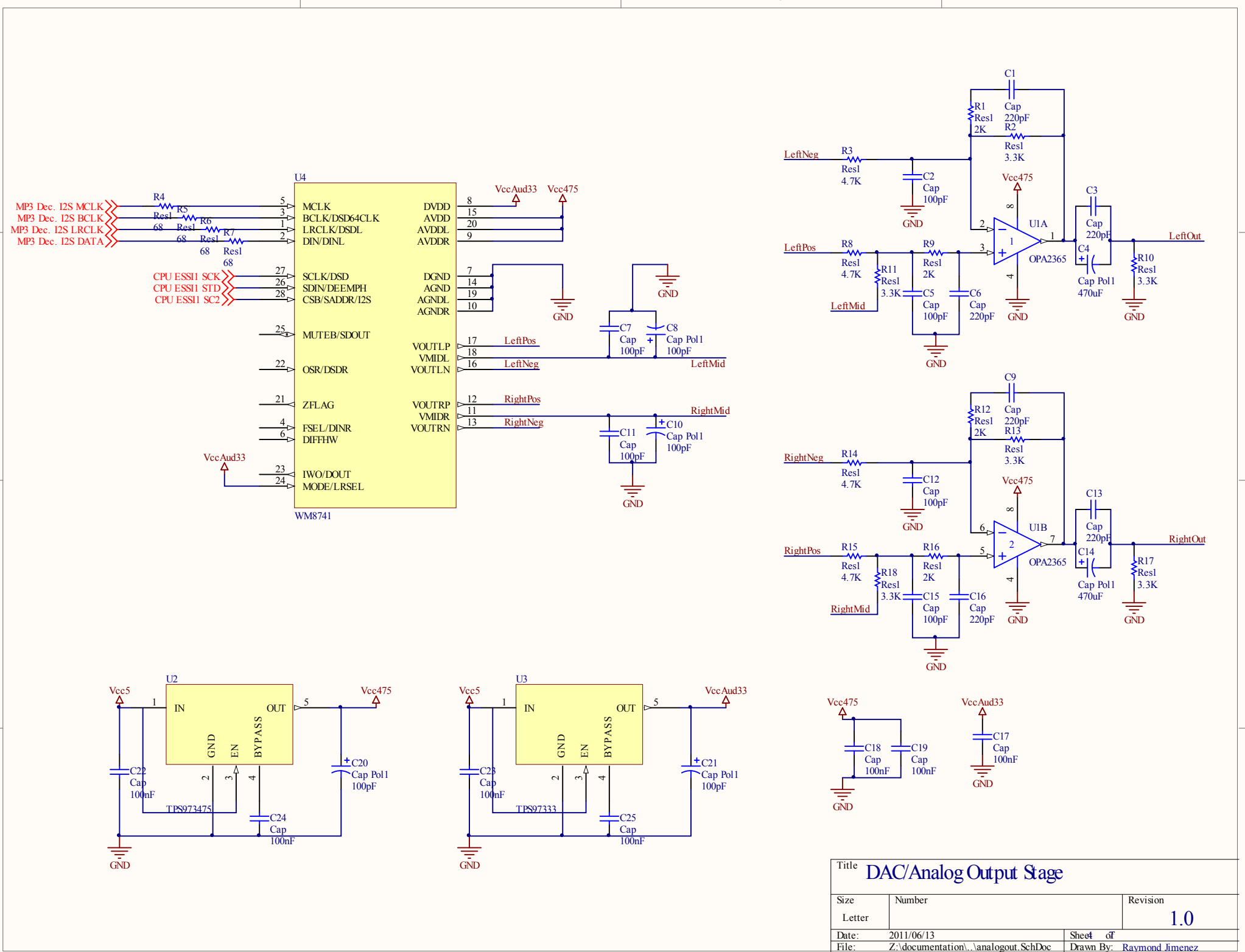
Title		
CPLD/Reset Logic		
Size	Number	Revision
Letter		1.0
Date:	2011/06/13	Sheet of
File:	Z:\documentation\...\cpld.SchDoc	Drawn By: Raymond Jimenez

7.3 MP3 Decoder Daughterboard



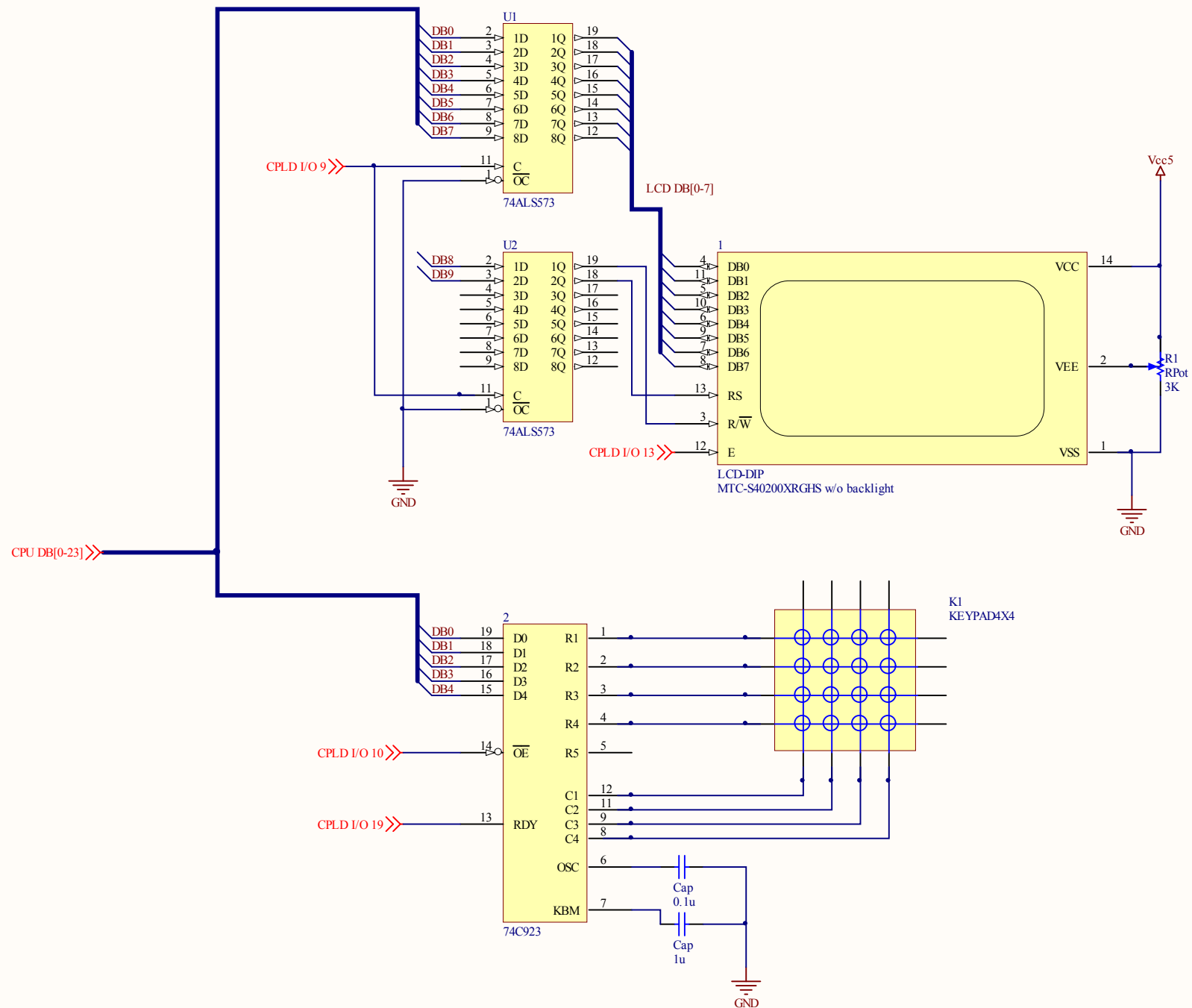
Title MP3 Decoder Daughterboard		
Size	Number	Revision
Letter		1.0
Date:	2011/06/13	Sheet of
File:	Z:\documentation\...\mp3daughter.SchDoc	Drawn By: Raymond Jimenez

7.4 DAC/Analog Out



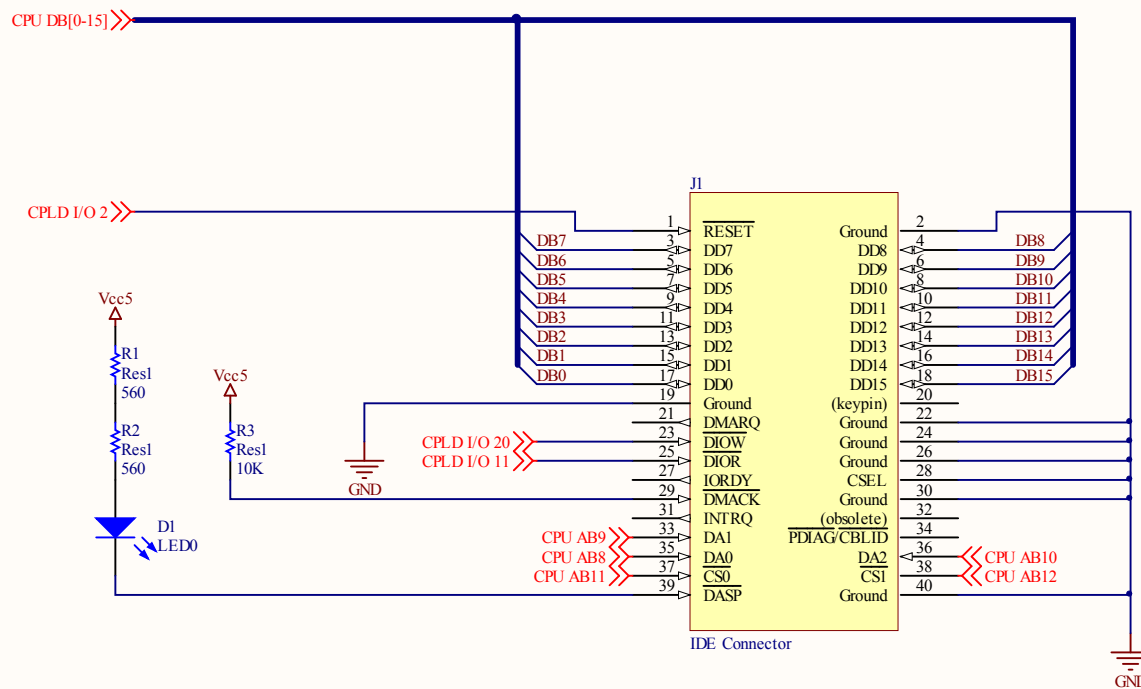
Title		
DAC/Analog Output Stage		
Size	Number	Revision
Letter		1.0
Date:	2011/06/13	Sheet of
File:	Z:\documentation\analogout.SchDoc	Drawn By: Raymond Jimenez

7.5 Display/Keypad



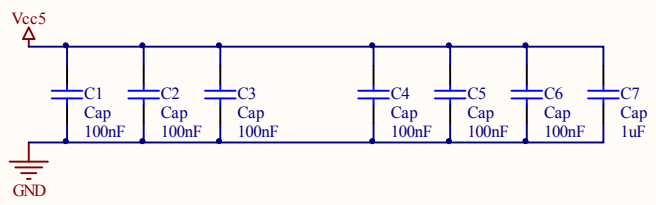
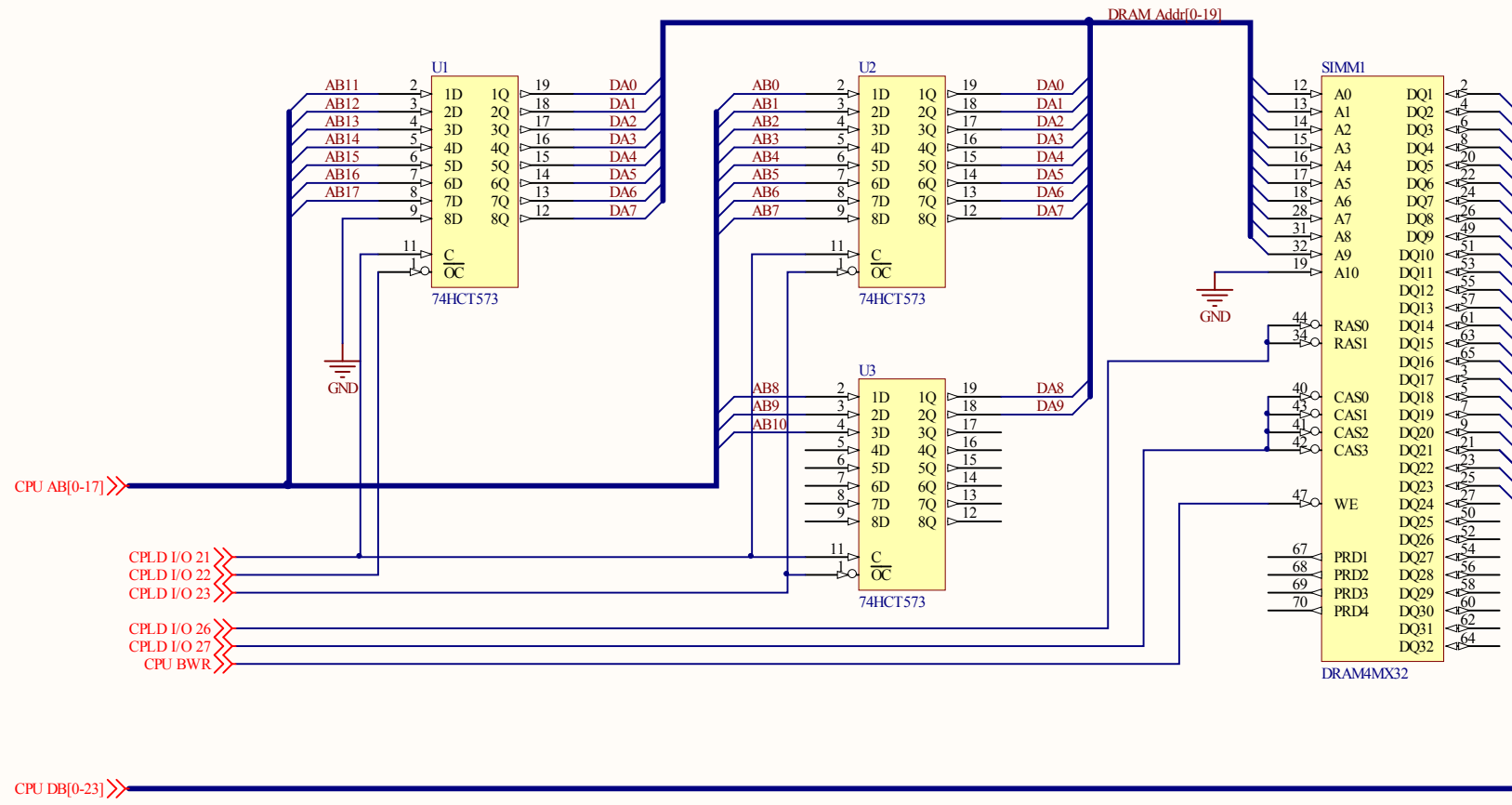
Title LCD Display/Keypad		
Size	Number	Revision
Letter		1.0
Date:	2011/06/13	Sheet of
File:	Z:\documentation\...\display.SchDoc	Drawn By: Raymond Jimenez

7.6 IDE Interface



Title IDE Interface		
Size	Number	Revision
Letter		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 of
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 hierarchical 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 necessary, 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_0/~
// 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.

```

1  ; crt0.asm
2  ;
3  ; This file is the initial bootup code that's executed from ROM.
4  ; This file is heavily adopted from the standard crt0.asm
5  ; that was included with the DSP56k developer toolkit.
6  ;
7  ; Perhaps most importantly, this file also incorporates the memory map
8  ; in order to prevent the compiler from storing data in prohibited
9  ; areas.
10 ;
11 ; Raymond Jimenez
12 ; EE/CS 52
13 ; February 16, 2011
14 ;
15 ; Revision history:
16 ;
17 ; 2011/2/16      Raymond Jimenez      initial revision
18 ;
19
20     opt      noidw
21     opt so,xr
22     page 132,66,3,3
23
24
25     ;      This section should be loaded at P:0 that way reset will
26     ;      cause a jump to main.
27
28 ;      The reset section is located at p:0 in the mapfile. This section is
29 ;      intended to hold system reset code. This code should jump to start
30 ;      when it is done.
31
32     section reset
33
34     org      p:$0
35     bra      >F__start
36
37 ; see http://vnsnes.freeshell.org/dsp/c\_intro.html,
38 ; we change this to 127 in order to duplicate over all 128 event handlers
39     org      p:$2
40     dup      127
41     bsr      >Fabort
42     endm
43
44     endsec
45
46     section crt0
47
48     org      x:
49
50 ; #####
51 ; The following variables are used for dynamic memory allocation
52
53 ; __stack_saftey: Since dynamic memory and the stack grow towards each other
54 ; This constant tells brk and sbrk what the minimum amount of space should
55 ; be left between the top of stack during the brk or sbrk call and the end
56 ; of any allocated memory.
57

```

```

58     global F__stack_safety
59 F__stack_safety dc      1024
60
61 TOP_OF_MEMORY equ      $a07fff
62     ;this is the top of our memory in X space; anything below this is SRAM
63     ; and is writable
64 BOTTOM_OF_MEMORY equ      $a00000
65     ;and then DSIZE is the bottom of our memory in X space.
66
67 ; __mem_limit: a constant telling brk and sbrk where the end of available
68 ;     memory is.
69
70     global F__mem_limit
71 F__mem_limit dc      TOP_OF_MEMORY
72
73
74 ; __break: pointer to the next block of memory that can be allocated
75 ;     The heap may be moved by changing the initial value of __break.
76 ;     This is the base of the heap.
77
78     global F__break
79 F__break dc      TOP_OF_MEMORY
80
81 ; __y_size: the base of dynamic memory.
82
83     global F__y_size
84 F__y_size dc      BOTTOM_OF_MEMORY
85
86 ; errno: error type: set by some libraries
87
88     global Ferrno
89 Ferrno dc      $0
90
91 ; __max_signal the highest possible signal vector offset that might
92 ;     be generated by the cpu.
93 ; see http://vnsnes.freeshell.org/dsp/c\_intro.html,
94 ; we change this to $fe in order to get all the possible interrupts
95
96     global F__max_signal
97 F__max_signal dc      $fe
98
99
100 ; #####
101
102     org      p:
103
104     global F__start
105 F__start
106
107 ;we do our initialization here before anything else happens
108 ; specifically: setup clock freq and chip selects
109
110 ;setup PLL and clock multiplier
111     movep #$040003,x:$fffffd ; 4 * 20.000 = 80.000MHz
112
113 ;map external memories per memory map
114     movep #$100431,x:$fffff9 ; AA0 = general periphs

```

```

115         movep #$d00609,x:$ffffff8 ; AA1 = ROM (forced due to mode 9)
116         movep #$200431,x:$ffffff7 ; AA2 = DRAM (via our own controller)
117         movep #$a00831,x:$ffffff6 ; AA3 = SRAM
118
119         jmp     end_xload
120
121 ;now that we can access the ROM, unpack x-space constants down there
122 ;judiciously adapted from the bootcode of the DSP56309 per the user manual
123
124         move   #$5400,a0           ;we fill all of x-space internals: $1bff * 3
125         move   #$0,a1             ;we load at the bottom of x-space
126         move   a1,r0              ; starting address for load
127         move   a1,r1              ; save it in r1
128                                     ; a0 holds the number of words
129         move   #$d20000,r2        ;r2 is the base where we read from
130         do a0,_LOOP10             ; read program words
131         do #3,_LOOP11            ; Each instruction has 3 bytes
132         movem p:(r2)+,a2          ; Get the 8 LSB from ext. P mem.
133         asr    #8,a,a             ; Shift 8 bit data into A1
134 _LOOP11                          ; Go get another byte.
135         move   a1,x:(r0)+         ; Store 24-bit result in X mem.
136         nop                      ; pipeline delay
137 _LOOP10                          ; and go get another 24-bit word.
138                                     ; x-space load from EPROM done
139 end_xload:
140
141 ;now set bus control register to set wait states
142         movep  #$01d27f,x:$fffffb ; see memory map
143
144 ;setup the stack extensions before we do anything
145         movec  #$000000,EP         ;set extension to start at bottom of intern. X
146         movec  #$00008f,SZ        ;and we have 256 words allocated
147
148         movec  #$c00300,SR        ;we set the SR to mask interrupts; main will
149                                     ; take care of unmasking them
150         movec  #$100809,OMR       ;and set the OMR to enable TA sync and mode 9
151                                     ; as well as stack extensions in X space
152
153         movec  #$000000,EP         ;set extension to start at bottom of intern. X
154         movec  #$00008f,SZ        ;and we have 256 words allocated
155
156
157
158 ;      To change the base of the stack, change the value loaded into the
159 ;      stack pointer here.
160
161         and    #$f3,mr
162         move   x:F__y_size,r6      ; initialize the stack pointer
163
164         bsr    F__init_c_vars
165
166         bsr    Fmain               ; run user program
167
168         bsr    Fexit               ; when exit ( ) isn't explicitly called,
169                                     ; call it.
170         global F__crt0_end
171 F__crt0_end

```

```

172         stop                ; all done
173
174     endsec
175
176     section time_counter
177
178         org      x:
179         global   F__time
180 F__time
181         dc       0
182
183     endsec
184
185     section io_primitives
186
187         org      p:
188
189         nop
190         global   F__send
191 F__send
192         rts
193         nop
194
195         nop
196         global   F__receive
197 F__receive
198         rts
199         nop
200
201     endsec
202
203 ; The following section, dummy_call, is used for doing command-line
204 ; function calls on the ADS and simulator.  If you aren't going to use
205 ; those features, it can be removed.
206
207 ; The host software sets up the dummy call and patches the following
208 ; object code to do the call and to clean things up after the call.
209 ; If the function that the user is calling from the command line
210 ; returns a structure or union, the value of r6 is moved into r7, and
211 ; then r6 is incremented by the size of the structure or union being
212 ; returned.  This negative of this size is then patched into the
213 ; second ``move #N,n6'' instruction, so that we can clean up after the
214 ; function call.  If no structure is returned, 0 is patched in for N
215 ; in that move.  Following this, the current PC is moved into x:(r6)
216 ; and then r6 is incremented.  The arguments to the function are then
217 ; placed into registers A and B, with the remainders placed on the
218 ; stack starting at r6 (with r6 being incremented accordingly).
219 ; The size of the arguments plus one (for the storage of the current
220 ; PC) will needed to be cleaned off the stack after the function call,
221 ; so we negate that value and patch it into the first ``move #N,n6''
222 ; instruction.  The address of the target function is then patched
223 ; into the jsr instruction.
224 ;
225 ; Following this the device is started and a ``finish'' (essentially)
226 ; is done on dummy_call (which will then call the target function).
227 ; If the function call is interrupted by the host software, the
228 ; ADS/simulator is left in a state where the user can step out of the

```

```

229 ; called function (because of the prologue code which is after the jsr
230 ; in dummy_call).
231 ;
232 ; The prologue code decrements r6 by the amount used by the arguments
233 ; to the target function plus one (to move us back to where the return
234 ; address was stored). Then x:(r6) is moved into ssh. Then a (the
235 ; return value) is tested, and r6 is decremented by the size of the
236 ; structure returned from the target function (possibly 0 if no
237 ; structure was returned). Finally, we return to our caller.

```

```

238
239     section dummy_call
240
241     org     p:
242
243     global  dummy_call
244 dummy_call
245     jsr     >dummy_call
246     move    #-1,n6
247     move    (r6)+n6
248     move    #-1,n6
249     move    x:(r6),ssh
250     tst     a      (r6)+n6
251     rts
252
253     endsec
254
255     section init_table
256     org p:
257     xdef     F__begin_init_table
258 F__begin_init_table
259     dc (-1)
260
261     endsec
262
263     section cinit
264     org     p:
265     xref     F__begin_init_table
266     global   F__init_c_vars
267 F__init_c_vars
268     clr     a
269     clr     b
270     move    #$ffffff,m0
271     move    m0,m1
272     move    m0,m2
273     move    #F__begin_init_table,r0
274
275     do      forever,_lab1
276 ; p:(r0)    destination start
277 ; p:(r0+1)  source start
278 ; p:(r0+2)  size of source block size
279 ; p:(r0+3)  size zero initialized block
280
281     movem   p:(r0)+,r1      ; destination start
282     lua     (r1)+,b
283     tst     b
284     nop
285     brkeq

```



```

286         movem p:(r0)+,r2           ; source start
287         movem p:(r0)+,r3           ; size of explicitly initialized block
288         movem p:(r0)+,r4           ; size of zero initialized block
289
290         do    r3,_lab2
291         movem p:(r2)+,x0
292         move  x0,x:(r1)+
293         nop
294 _lab2
295
296         do    r4,_lab3
297         move  a1,x:(r1)+
298 _lab3
299         nop
300 _lab1
301         rts
302
303         endsec
304
305 ;
306 ; we reserve memory space here to prevent things from going awry.
307 ; i.e., we block it off to the compiler to make it seem like there
308 ; is already something using this non-existent memory.
309 ;
310 ; this was done in a much nicer fashion until I accidentally deleted
311 ; this file; you'll have to live with larger granularity now.
312 ;
313 ; please refer to the memory map documentation for more details.
314 ;
315
316 ; Program space reservations
317
318 ; reserve everything above the internal 20K program RAM,
319 ; nothing else is writeable in P space
320     section p_memory_reserved
321     org    p:$005000           ;starts at 0x005000
322     ds     $FFB000             ;0x1000000 - 0x005000
323     endsec
324
325 ; X space reservations
326
327 ;cover the peripheral memory and unaddressed memory gaps
328 ; (also includes memory-mapped registers)
329     section x_memory_reserved0
330     org    x:$a10000           ;starts above the SRAM
331     ds     $5f0000             ;0x1000000 - 0xa10000
332     endsec
333
334 ;cover half of sram for testing purposes (fake dram)
335     section sram_temp_res
336     org    x:$a08000           ;starts in the middle of the SRAM
337     ds     $8000               ;0xa10000-0xa08000
338     endsec
339
340 ;cover between sram and dram
341     section x_memory_reserved1
342     org    x:$300000           ;starts above the DRAM

```

```
343         ds      $700000          ;0xa00000 - 0x300000
344     endsec
345
346 ;cover the DRAM for now because it is not functional
347     section dram_temp_reservation
348     org      x:$200000          ;starts at 0x200000
349     ds      $100000          ;0x300000 - 0x200000 (1MByte ostensibly)
350     endsec
351
352 ;cover the gap between DRAM and peripherals
353     section x_memory_reserved2
354     org      x:$104000          ;starts at 0x104000
355     ds      $fc000          ;0x200000 - 0x104000
356     endsec
357
358 ;we can't actually store anything in peripheral memory, so
359 ; block it off.
360     section gen_periph
361     org      x:$100000          ;starts at 0x100000
362     ds      $4000          ;0x104000 - 0x100000
363     endsec
364
365 ;cover the gap between peripherals and internal X ram
366     section x_memory_reserved3
367     org      x:$001c00          ;starts at 0x001c00
368     ds      $fe400          ;0x100000 - 0x001c00
369     endsec
370
371 ;we have 256 words of stack extension at the beginning of
372 ; X space (internal SRAM), so block it off:
373     section x_mem_stack_ext
374     org      x:$000000          ;at the bottom of internal X RAM
375     ds      $100          ;256 words
376     endsec
377
```

8.3 Queues (**queues.asm**)

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

```

1
2 ; queues.asm
3 ;
4 ; This file offers several rudimentary queue (FIFO) functions.
5 ; Note that none of these functions follow the C calling convention;
6 ; they are ASM-call safe only.
7 ;
8 ; Many of these functions are ported from EE51 HW#3.
9 ;
10 ; queue_init - initializes a new queue; doesn't clear memory.
11 ; queue_empty - checks if a queue is empty
12 ; queue_full - checks if a queue is full
13 ; enqueue - adds a word to the back of the queue
14 ; dequeue - grabs the word from the front of the queue
15 ;
16
17 ; Raymond Jimenez
18 ; EE/CS 52
19 ; February 16, 2011
20 ;
21 ; Revision history:
22 ;
23 ; 2011/2/16      Raymond Jimenez      initial revision
24 ;
25
26
27 ; uh, from what I see, the DSP assembler does not provide a struct construct.
28 ; as a result, here's what we expect when we see a queue pointer:
29 ;
30 ; 24          0
31 ; +-----+ 0
32 ; | head pointer |
33 ; |-----| 1
34 ; | tail pointer |
35 ; |-----| 2
36 ; | size (length of
37 ; |   data section) |
38 ; |-----| 3
39 ; |   .           |
40 ; |   .           |
41 ; |   .           |
42 ; |   data        |
43 ; |   .           |
44 ; |   .           |
45 ; |   .           |
46 ; +-----+ 3+size
47 ;
48 ;
49         section queues
50 INCLUDE 'macros.inc'
51
52         org      p:      ;we want to be located in program space to be in ROM
53
54
55 ; queue_init
56 ; Functional Specification
57 ;

```

```

58 ; Description:      queue_init zeros out the queue structure so that
59 ; it can be used by other queue functions later. we assume all queues
60 ; are in X space.
61 ;
62 ; Operation:      queue_init sets the head and tail pointers to be
63 ; the same (0) and sets the size to be the passed size.
64 ;
65 ; Arguments:      r0 - address of queue (in X space)
66 ;                 r1 - size of queue (total number of items - 1)
67 ;
68 ; Return Values:  None.
69 ;
70 ; Global Variables: None.
71 ;
72 ; Shared Variables: None.
73 ;
74 ; Local Variables: None.
75 ;
76 ; Inputs:         None.
77 ;
78 ; Outputs: None.
79 ;
80 ; Error Handling:  None.
81 ;
82 ; Algorithm:      None.
83 ;
84 ; Data Structures: None.
85 ;
86 ; Registers changed: None.
87 ;
88 ; Limitations:    None known.
89 ;
90 ; Known Bugs:     None.
91 ;
92 ; Special Notes:
93 ;
94 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
95 ;
96     global queue_init    ;prologue
97 queue_init:              ;assembly entry point
98
99     ;save registers we use:
100    move    r3,ssh
101
102    move    #0,r3          ;use r3 temporarily to hold 0
103    move    r3,x:(r0)       ;set the head pointer to 0
104    move    r3,x:(r0+1)     ;set the tail pointer to 0
105    move    r1,x:(r0+2)     ;and set the size in size field
106
107    ;restore registers we used:
108    move    ssh,r3
109
110    rts    ;return from subroutine
111
112 ; queue_empty
113 ; Functional Specification
114 ;

```

```

115 ; Description:      queue_empty checks whether the queue is empty, by comparing
116 ; the head and tail pointers, and sets ZF if the queue is empty.
117 ;
118 ; Operation:        The function compares the head and tail pointers, and as a
119 ; result, sets the zero flag. In other words, the cmp instruction does all of
120 ; the work for us.
121 ;
122 ; Arguments:         r0 - address of the queue to look at
123 ;
124 ; Return Values:     ZF - set if queue is empty
125 ;
126 ; Global Variables:  None.
127 ;
128 ; Shared Variables:  None.
129 ;
130 ; Local Variables:   None.
131 ;
132 ; Inputs:            None.
133 ;
134 ; Outputs:           None.
135 ;
136 ; Error Handling:    None.
137 ;
138 ; Algorithm:         None.
139 ;
140 ; Data Structures:   None.
141 ;
142 ; Registers changed:  None.
143 ;
144 ; Limitations:       None known.
145 ;
146 ; Known Bugs:        None.
147 ;
148 ; Special Notes:
149 ;
150 ; Revision History:  2011/02/16    Raymond Jimenez    first revision
151 ;
152     global queue_empty
153 queue_empty:                ;assembly entry point
154
155     ;we use the accumulators due to CMP, so save them:
156     PUSH_ACC A
157     move    x0,ssh          ;we'll be nice and use only x0 in addition to a
158
159     move    x:(r0),a        ;move head pointer into accumulator
160     move    x:(r0+1),x0     ;move tail pointer into x0 for CMP
161
162     cmp     x0,a            ;test to see if they are the same
163                               ; (this also sets ZF)
164
165     move    ssh,x0          ;pop x0 back off the stack
166     POP_ACC A
167
168
169     rts                ;return from subroutine
170
171

```

```

172 ; queue_full
173 ; Functional Specification
174 ;
175 ; Description:      queue_full takes the head and tail pointers, does
176 ; a little bit of modulo arithmetic to increment the tail pointer, and tells
177 ; us whether the queue is full or not.
178 ;
179 ; Operation:        The function gets the head and tail pointers, and increments
180 ; the tail pointer. If the tail pointer points to an invalid slot, it's wrapped
181 ; back to the first slot. After this, the two pointers are compared; if they are
182 ; equal (the queue is full), then the ZF flag is set, otherwise unset.
183 ;
184 ; Arguments:        r0 - address of the queue to look at
185 ;
186 ; Return Values:    ZF - set if queue is full
187 ;
188 ; Global Variables: None.
189 ;
190 ; Shared Variables: None.
191 ;
192 ; Local Variables:  None.
193 ;
194 ; Inputs:           None.
195 ;
196 ; Outputs:          None.
197 ;
198 ; Error Handling:   None.
199 ;
200 ; Algorithm:        None.
201 ;
202 ; Data Structures:  None.
203 ;
204 ; Registers changed: None.
205 ;
206 ; Limitations:      None known.
207 ;
208 ; Known Bugs:       None.
209 ;
210 ; Special Notes:
211 ;
212 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
213 ;
214     global queue_full
215 queue_full:                ;assembly entry point
216
217     ;we use the accumulators due to CMP/SUB, so save them:
218     PUSH_ACC A
219     ;save x0, x1
220     move    x0,ssh
221     move    x1,ssl          ;and then x1 into ssl
222
223     move    x:(r0),x0       ;move head pointer into x0
224     move    x:(r0+1),a      ;move tail pointer into accumulator for possible
225
226                             ; SUB
227     move    x:(r0+2),x1     ;move size into x1 so we can compare it

```

```

228         add     #>1,a           ;we increment the pointer to the next address
229
230         cmp     x1,a             ;we see if the pointer is too large (x1<a)
231         jle     queue_full_nowrap
232                                     ;if tail >= size, wrap to zero
233         ;jgt     queue_full_wrap
234 queue_full_wrap:
235         clr     a                 ;we reset our tail pointer to 0 to wrap
236
237 queue_full_nowrap:
238
239         cmp     x0,a             ;test to see if they are the same
240                                     ; (this also sets ZF)
241
242         move     ssl,x1           ;pop x1 back off the stack
243         move     ssh,x0           ;restore x0
244         POP_ACC A
245
246
247         rts     ;return from subroutine
248
249 ; dequeue
250 ; Functional Specification
251 ;
252 ; Description:      dequeue takes a queue and returns the first object
253 ; of the queue. it blocks if there is nothing in the queue to return.
254 ;
255 ; Operation:        Dequeue takes the queue and calls queue_full to see
256 ; if it is full; if so, it waits for it to dequeue. once that is checked,
257 ; it then puts the value of the item in a0, and increments the head
258 ; pointer. if this pointer is too large (>=size) it is reset back to zero,
259 ; to wrap to the first element.
260 ;
261 ; all of this is wrapped in critical code statements because it is
262 ; ill-defined what a queue should do when interrupted in this process.
263 ;
264 ; Arguments:        r0 - the address of the queue to dequeue
265 ;
266 ; Return Values:    a1 - the value of the item at the front of the queue
267 ;                  a0 is cleared, a2 sign-extended as necessary.
268 ;
269 ; Global Variables: None.
270 ;
271 ; Shared Variables: None.
272 ;
273 ; Local Variables: None.
274 ;
275 ; Inputs:           None.
276 ;
277 ; Outputs:          None.
278 ;
279 ; Error Handling:   None.
280 ;
281 ; Algorithm:        None.
282 ;
283 ; Data Structures: None.
284 ;

```



```

285 ; Registers changed: a2, a1, a0 (a)
286 ;
287 ; Limitations:      None known.
288 ;
289 ; Known Bugs:      None.
290 ;
291 ; Special Notes:
292 ;
293 ; Revision History: 2011/02/16      Raymond Jimenez      first revision
294 ;
295     global  dequeue
296 dequeue:                                     ;assembly entry point
297
298     ;we use the accumulators due to CMP/INC, so save them:
299     PUSH_ACC b
300
301     ;and now save x0 & n0
302     move    x0,ssh
303     move    n0,ssl
304
305     ;check if there are no objects to dequeue
306 dequeue_block_loop:
307     jsr     queue_empty
308     jeq     dequeue_block_loop      ;if there is nothing to dequeue, block.
309
310     ;since we grab the head pointer, from here on it is critical code
311     START_CRITICAL
312     ;-- START OF CRITICAL CODE --
313
314     move    x:(r0)+,b               ;grab the head pointer so that we can modify it
315                                         ;(and do comparisons, increments on it)
316     move    b1,n0                   ;also put it in n0 so we can grab the item
317     move    x:(r0+1),x0             ;grab size for wrap usage
318
319     move    (r0)+                   ;increment pointer into data space
320     move    (r0)+
321     move    x:(r0+n0),a             ;grab our item into a
322
323     ;we take care of wrapping, now
324
325     add     #1,b                    ;increment the head pointer
326     cmp     x0,b                    ;is the head pointer too large?
327     jle     dequeue_no_wrap         ;do not wrap if the size is larger than the
                                     ;pointer
328     ; jgt     dequeue_wrap
329 dequeue_wrap:
330     clr     b                       ;we clear the head pointer.
331
332 dequeue_no_wrap:
333     move    (r0)-                   ;decrement pointer to start of queue
334     move    (r0)-
335     move    (r0)-
336     move    b1,x:(r0)               ;replace the head pointer
337
338     ; -- END OF CRITICAL CODE --
339     END_CRITICAL
340

```

```

341
342     ;end of body code, now clean up registers
343     move    ssl,n0           ;restore n0/x0
344     move    ssh,x0
345
346     POP_ACC b
347
348     rts      ;return from subroutine
349     ;end of dequeue
350
351
352 ; enqueue
353 ; Functional Specification
354 ;
355 ; Description:      enqueue takes a single argument in a0 to add to the end of
356 ; the queue. it places it at the end of the queue; if there is no space, it
357 ; will block until there is space.
358 ;
359 ; Operation:        enqueue first checks if there is space left in the queue .
360 ; if there is no space, it will block until there is space. it then places the
361 ; item at the end of the queue, and then increments the pointer and wraps it
362 ; as necessary to fit within the data block.
363 ;
364 ; Arguments:        a1 - the data to place at the end of the queue
365 ;                   (we don't use a0/a2)
366 ;                   r0 - the address of the queue to update
367 ;
368 ; Return Values:    None.
369 ;
370 ; Global Variables: None.
371 ;
372 ; Shared Variables: None.
373 ;
374 ; Local Variables:  None.
375 ;
376 ; Inputs:           None.
377 ;
378 ; Outputs:          None.
379 ;
380 ; Error Handling:    None.
381 ;
382 ; Algorithm:         None.
383 ;
384 ; Data Structures:  None.
385 ;
386 ; Registers changed: None.
387 ;
388 ; Limitations:       None known.
389 ;
390 ; Known Bugs:        None.
391 ;
392 ; Special Notes: much of this code is ill-defined as to what to do when
393 ; interrupted, so we block of interrupts for the update portion.
394 ;
395 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
396 ;
397     global  enqueue

```

```

398 enqueue:                                ;assembly entry point
399
400     ;save all registers we use so we can restore later
401     ;we use the accumulators due to CMP/SUB, so save them:
402     PUSH_ACC A
403     ;then we used n0/x0, so save them:
404     move    x0,ssh
405     move    n0,ssl
406
407
408     ;check if there is any space left to queue
409 enqueue_block_loop:
410     jsr     queue_full    ;check if queue is full
411     jeq     enqueue_block_loop    ;if so, loop
412
413
414     ;since we grab the tail pointer, from here on it is critical code
415     START_CRITICAL
416     ;-- START OF CRITICAL CODE --
417
418     move    x:(r0+1),n0    ;we will use the tail pointer as an offset
419
420     ;we put the object into the queue:
421     move    (r0)+          ;increment the pointer to the beginning of data
422     move    (r0)+
423     move    (r0)+
424     move    a1,x:(r0+n0)    ;grab from a0 and put it into the queue
425     move    (r0)-          ;and restore r0 to the start of the queue struct
426
427     move    (r0)-
428
429     ;and now we worry about wrapping the tail pointer.
430
431     move    x:(r0+1),a      ;use the a accumulator for tests etc
432     add     #>1,a          ;increment the tail pointer
433     move    x:(r0+2),x0     ;grab the size into x0 so we can compare with it
434
435     cmp     x0,a           ;compare size and tail pointer
436     jle     enqueue_no_wrap ;do not wrap if size>pointer
437     ;jgt     enqueue_wrap
438 enqueue_wrap:
439     clr     a              ;reset tail pointer to 0
440
441 enqueue_no_wrap:
442     move    a1,x:(r0+1)    ;update tail pointer in the queue
443
444
445     ; -- END OF CRITICAL CODE --
446     END_CRITICAL
447
448     ;end of body code, now restore all registers
449     move    ssl,n0        ;restore n0/x0
450     move    ssh,x0
451
452     POP_ACC A

```

```
453
454         rts      ;return from subroutine
455 ;end function
456
457
458
459
460         endsec
461
462
463
```

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.

```

1 ; display.inc
2 ;
3 ; This file offers several key includes for the display functionality,
4 ; such as size of memory buffers and constants, etc.
5 ;
6 ; Revision History:
7 ;
8 ; 2011/2/15      Raymond Jimenez      initial revision
9 ; 2011/6/12      Raymond Jimenez      cleaned up for documentation
10 ;
11 ;
12
13
14 ;allocate 256 words for the LCD queue
15 DISPLAY_QUEUE_SIZE      EQU      $200
16
17
18 ;our display is located at 0x100000
19 DISPLAY_ADDR            EQU      $100000
20
21 ;allocate 255 characters to be written to the LCD
22 ; at any given time.
23 DISPLAY_TEMP_BUF        EQU      $ff
24
25
26 ;we steal these from interfac.h so that we can
27 ; use them in assembly.
28
29 STATUS_PLAY              EQU      0
30 STATUS_FASTFWD           EQU      1
31 STATUS_REVERSE           EQU      2
32 STATUS_IDLE              EQU      3
33 STATUS_ILLEGAL           EQU      4
34
35 TIME_NONE                EQU      65535
36
37 ;           our display looks like the following:
38 ;           1           2           3           4
39 ;           1234567890123456789012345678901234567890
40 ;           | artist                                |> Play -127dB
41 ;           | title                                | time   |
42 ;
43
44
45 STATUS_ADDR              EQU      26
46 ATTEN_ADDR               EQU      34
47
48 ATTEN_SHIFT              EQU      3 ;shift over 3 to get full integer dB
49

```

```

1  ; display.asm
2  ;
3  ; This file contains the hardware-level interface code for the LCD display.
4  ; It offers the functions:
5  ;
6  ; void display_time(unsigned int) - display the passed time on the LCD
7  ; void display_status(unsigned int) - display the passed status on the LCD
8  ; void display_title(char far *) - display the title on the LCD
9  ; void display_artist(char far *) - displays the artist on the LCD
10 ;
11 ; void display_update(void) -- updates the LCD; should be called from a timer
12 ; interrupt
13 ;
14 ; void display_init(void) - initializes the LCD structures and hardware; should
15 ; be called before any other display calls are made
16 ;
17 ; Revision History:
18 ;
19 ; 2011/2/15      Raymond Jimenez      initial revision
20 ; 2011/6/12      Raymond Jimenez      cleaned up for documentation
21 ;
22 ;
23
24
25     section display      ;here's our declaration for our file
26 INCLUDE 'display.inc'
27 INCLUDE 'macros.inc'
28
29     ;we declare volatile data structures first
30     org x:                ;keep all the large stuff in sram/dram/x ram
31
32     ; VOLATILE DEFINITIONS START HERE
33
34 display_queue:
35     ds $1                ;keep a single 24-bit counter around for the
36                           ; head offset
37     ds $1                ;and the tail offset
38
39     ds DISPLAY_QUEUE_SIZE ;queue data
40
41     ; we locate several string constants here to simplify our life.
42     ; we simply assume all strings passed to our writing functions
43     ; are located in x-space; for C-passed strings this is certainly
44     ; true (many C constants/data structures are located in X space)
45     ; so we adopt the same standard for strings.
46
47     ; our display looks like the following:
48     ;           1           2           3           4
49     ; 1234567890123456789012345678901234567890
50     ; | artist                |> Play -127dB
51     ; | title                 | time   |
52     ;
53
54
55     ;several fields need to be cleared every time they are used,
56     ; like the title and artist fields. we assume a 2x40 LCD.
57     ; we have:

```

```

58
59 clear_artist_str:
60 ;           1           2           3           4
61 ;           1234567890123456789012345678901234567890
62         dc "                                ", $00
63 clear_title_str:
64 ;           1           2           3           4
65 ;           1234567890123456789012345678901234567890
66         dc "                                ", $00
67 clear_time_str:
68 ;           1           2           3           4
69 ;           1234567890123456789012345678901234567890
70         dc "                                ", $00
71
72 clear_vol_str:
73 ;           1           2           3           4
74 ;           1234567890123456789012345678901234567890
75         dc "                                ", $00
76
77 time_minute_sep:
78         dc ":", $00
79 time_tenths_sep:
80         dc ":", $00
81
82 atten_prefix:
83         dc "-", $00
84 atten_suffix:
85         dc "dB", $00
86
87         ;we have several constants corresponding to what to display
88         ; for a given status:
89 status_play_str:
90         dc "|> Play ", $00
91 status_fastfwd_str:
92         dc ">> FF  ", $00
93 status_reverse_str:
94         dc "<< Rev ", $00
95 status_idle_str:
96         dc "[ ] Stop ", $00
97 status_illegal_str:
98         dc "OTL   ", $00
99
100         ;we require a temporary buffer to place our results from
101         ; dec2string.
102 convert_temp_buf:
103         ds $ff           ; we allocate 255 words, which will inevitably be
104                           ; 255 right-aligned ASCII characters.
105         dc $00           ; and we null terminate, in case.
106
107         ;and from here, we declare things to be burned in ROM
108         org p:
109
110         ; CONSTANT DEFINITIONS START HERE
111
112         ;we have a lookup table for all of the statuses:
113 status_str_table:
114         dc status_play_str           ;0 - play

```



```

115         dc status_fastfwd_str          ;1 - fastfwd
116         dc status_reverse_str          ;2 - reverse
117         dc status_idle_str             ;3 - idle
118         dc status_illegal_str          ;4 - illegal
119
120         ; PROGRAM CODE STARTS HERE
121
122         ;and finally, program code
123         global display_update
124 display_update:                          ;assembly entry only
125         ; we are here from an interrupt, so be quick and careful
126         ; do not worry about clearing the interrupt pin just yet
127         move    r0,ssh                  ;put r0 on stack; we can't modify it
128         move    r1,ssl
129
130         ;put a on stack, must keep it clean
131         PUSH_ACC a
132
133         ;Save all registers to be used.
134         move    #display_queue,r0      ;setup for calling dequeue/queue empty
135
136         ;check if there's any commands to be sent
137         jsr     queue_empty
138         jeq     display_update_done    ;if empty, don't do anything
139
140 display_update_process:
141         jsr     dequeue                  ;else, we dequeue the command and write
142
143         move    a1,x:DISPLAY_ADDR      ;and then write to the display
144
145
146
147 display_update_done:
148
149         ;Restore all registers used.
150         ;restore a accumulator first
151         POP_ACC a
152
153         move    ssl,r1
154         move    ssh,r0                  ;pop r0 back off of stack
155
156
157         rts
158
159 ; Fdisplay_init
160 ; Functional Specification
161 ;
162 ; Description:      Fdisplay_init ( void display_init() ) should be called
163 ; from a C calling function to initialize the display mechanics, such as
164 ; queues etc.
165 ;
166 ; Operation:       Fdisplay_init initializes the display command queue
167 ; using queue_init, but that's it.
168 ;
169 ; Arguments:       None.
170 ;
171 ; Return Values:   None.

```

```

172 ;
173 ; Global Variables: None.
174 ;
175 ; Shared Variables: None.
176 ;
177 ; Local Variables: None.
178 ;
179 ; Inputs:           None.
180 ;
181 ; Outputs:          None.
182 ;
183 ; Error Handling:   None.
184 ;
185 ; Algorithm:        None.
186 ;
187 ; Data Structures: None.
188 ;
189 ; Registers changed: None.
190 ;
191 ; Limitations:      None known.
192 ;
193 ; Known Bugs:       None.
194 ;
195 ; Special Notes:
196 ;
197 ; Revision History: 2011/02/17   Raymond Jimenez   first revision
198 ;
199     global  Fdisplay_init
200 Fdisplay_init:                ;C entry point
201
202
203     ;standard C-caller prologue
204     move #0,n6                ; k is the amount of local space needed.
205     move ssh,x:(r6)+          ; save the return address.
206     move (r6)+n6              ; allocate local stack space of size k.
207
208     ;save all registers we use so we can restore later
209     move r0,ssh                ;we use r0 only
210     move r1,ssl                ;and r1 as well
211
212     move #display_queue,r0
213     move #DISPLAY_QUEUE_SIZE,r1
214     jsr  queue_init           ;init the display queue
215
216     ;we reset the display
217     move #>$1,a                ;we are going to reset the display hardware
218     jsr  enqueue              ;and enqueue it for the timer to take care of
219
220
221     ;however, this takes the display 1.53mS -- super long.
222     ; our timer is set to go every 50us, so we program in
223     ; 32 NOPs to allow the reset to proceed. (32*50us = 1.6mS)
224     move #>$0,a                ;do NOPs for the next instruction.
225     do   #32,display_init_loop_end          ;do 32 NOPs.
226
227     nop
228     jsr  enqueue

```

```

229         nop
230         nop
231
232 display_init_loop_end:
233
234
235         ;set our output mode: display on
236         move #>$0c,a
237         jsr  enqueue
238
239
240         move #>$38,a ;and make 2-line mode a thing.
241         jsr  enqueue ;and let the display interrupt take care of it
242
243         ;end of body code, now restore all registers
244
245         move ssl,r1      ;restore r1
246         move ssh,r0      ;r0 is now back to initial state
247
248         ;standard C-caller epilogue
249         move #0+1,n6
250         move (r6)-n6      ; deallocate local stack space, set ccr flags.
251         tst  a
252         move x:(r6),ssh ; get the return address.
253
254
255         rts      ;return from subroutine
256 ;end Fdisplay_init
257
258
259 ; display_write_string
260 ; Functional Specification
261 ;
262 ; Description:      This takes an ASCII null-terminated string and
263 ; writes it to the LCD display. The string must be in X space.
264 ;
265 ; Operation:       The function loops over the string, reading each
266 ; character at a time. If the string is a NULL ($0), it returns.
267 ; It takes the lowest byte of the word and enqueues it OR'd with
268 ; 0x000100, which is the R/W bit on the LCD (indicating a character).
269 ;
270 ; Arguments:       r0 - pointer to the string we want to enqueue
271 ;
272 ; Return Values:   None.
273 ;
274 ; Global Variables: None.
275 ;
276 ; Shared Variables: None.
277 ;
278 ; Local Variables: None.
279 ;
280 ; Inputs:         None.
281 ;
282 ; Outputs:        None.
283 ;
284 ; Error Handling:  None.
285 ;

```

```

286 ; Algorithm:      None.
287 ;
288 ; Data Structures: None.
289 ;
290 ; Registers changed: None.
291 ;
292 ; Limitations:     None known.
293 ;
294 ; Known Bugs:      None.
295 ;
296 ; Special Notes:
297 ;
298 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
299 ;
300     global display_write_string
301 display_write_string:                ;assembly entry point
302
303     ;save all registers we use so we can restore later
304
305     move    r0,ssh                    ;we save r0 since we'll alter it
306     move    r1,ssl                    ;as well as r1
307     ;we also use a0, so keep it on the stack
308     PUSH_ACC a
309
310     move    r0,r1                    ;we'll use r1 as the real counter
311     move    #display_queue,r0
312                                     ;and r0 pointing to the queue for
313                                     ; the enqueue call
314
315
316     ;initialize a with the first byte:
317     clr     a
318     move    x:(r1)+,a1
319
320     ;start the main loop:
321
322 write_string_loop:
323     nop
324
325     and     #>$7f,a                    ;make sure we have only ASCII chars
326     tst     a                        ;see if a is all zeros -> NULL
327     jeq     write_string_loop_end    ;if so, get out of here
328
329     or      #>$100,a                  ;place in the 'R/W' bit on the LCD
330     jsr     enqueue                  ;now enqueue it to be written
331
332     move    x:(r1)+,a1
333     nop
334     nop
335     nop
336     jmp     write_string_loop
337 write_string_loop_end:
338
339
340     nop
341     nop
342     nop

```

```

343
344
345     POP_ACC a
346     move    ssl,r1          ;restore r1
347     move    ssh,r0          ;restore r0
348
349
350     ;end of body code, now restore all registers
351     rts      ;return from subroutine
352 ;end display_write_string
353
354
355 ; display_set_loc
356 ;
357 ; Functional Specification
358 ;
359 ; Description:      display_set_loc allows one to control the location of
360 ; where a string will be blit onto the LCD display. For simplicity, it takes
361 ; a DD RAM address corresponding to a physical address on our 2x40 LCD.
362 ;
363 ; Row 1: $0-$27
364 ; Row 2: $40-67
365 ;
366 ; There is no error handling; if given an invalid address an unpredictable
367 ; result will occur.
368 ;
369 ; Operation:      display_set_loc enqueues the appropriate command bits
370 ; to set the LCD to the given address, and then relies on the interrupt/timing
371 ; routines to send the command to the LCD.
372 ;
373 ; Arguments:      a1 - DDRAM addresss to position the cursor at
374 ;
375 ; Return Values:   None.
376 ;
377 ; Global Variables: None.
378 ;
379 ; Shared Variables: None.
380 ;
381 ; Local Variables: None.
382 ;
383 ; Inputs:         None.
384 ;
385 ; Outputs:        None.
386 ;
387 ; Error Handling:  None.
388 ;
389 ; Algorithm:      None.
390 ;
391 ; Data Structures: None.
392 ;
393 ; Registers changed: None.
394 ;
395 ; Limitations:    None known.
396 ;
397 ; Known Bugs:     None.
398 ;
399 ; Special Notes:   Based on the MTC-S40200XRGHS datasheet.

```

```

400 ;
401 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
402 ;
403     global  display_set_loc
404 display_set_loc:                ;assembly entry point
405
406     ;save all registers we use so we can restore later
407     move    r0,ssh               ;we use r0 of course.
408     ;and we change the accumulator
409     PUSH_ACC a
410
411     ;alright, we alter a to include our nice bit-mask
412     ; that turns an address into an LCD command:
413     and     #>$7f,a             ;clear out anything above DB6
414     or      #>$80,a             ;DB7 needs to be high, nothing else
415
416     move    #display_queue,r0 ;get the address for enqueue
417     jsr     enqueue             ;enqueue the command
418
419     ;the LCD datasheet lies and needs a single nop after the
420     ; set location for timing
421     clr     a
422     jsr     enqueue
423
424     ;end of body code, now restore all registers
425     POP_ACC a
426     move    ssh,r0
427
428     rts     ;return from subroutine
429 ;end display_set_loc
430
431
432 ; Fdisplay_status
433 ; Functional Specification
434 ;
435 ; Description:      Fdisplay_status takes a status number and writes the
436 ; appropriate status to the display.
437 ;
438 ; Operation:        Fdisplay_status uses the status number as an offset
439 ; to lookup the appropriate string pointer in x-space, then calls
440 ; display_set_loc to set the location of the string, followed by a
441 ; call to display_write_string to actually blit the string to the display.
442 ;
443 ; Arguments:        a1 - status number to display, per interfac.h
444 ;
445 ; Return Values:    None.
446 ;
447 ; Global Variables: None.
448 ;
449 ; Shared Variables: None.
450 ;
451 ; Local Variables: None.
452 ;
453 ; Inputs:           None.
454 ;
455 ; Outputs:          None.
456 ;

```

```

457 ; Error Handling:  None.
458 ;
459 ; Algorithm:       None.
460 ;
461 ; Data Structures: None.
462 ;
463 ; Registers changed: None.
464 ;
465 ; Limitations:     None known.
466 ;
467 ; Known Bugs:      None.
468 ;
469 ; Special Notes:
470 ;
471 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
472 ;
473     global  Fdisplay_status
474 Fdisplay_status:                ;C entry point
475
476     ;standard C prologue
477     move    #0,n6                ; k is the amount of local space needed.
478     move    ssh,x:(r6)+          ; save the return address.
479     move    (r6)+n6              ; allocate local stack space of size k.
480
481     ;save all registers we use so we can restore later
482     move    r0,ssh
483     move    r1,ssl
484     move    n1,ssh
485
486     ;we also change b, so preserve it
487     PUSH_ACC b
488
489     ;we change the address to display in the upper top right corner, always.
490     ; since our strings are fixed length, we can just rely on them to
491     ; mask the previous status' text.
492
493     move    a,b                  ;keep our status around
494     move    #>STATUS_ADDR,a      ;we like to be in the top right corner.
495     jsr     display_set_loc
496     move    b,a
497
498     ;our argument should be in a1 because we only take a single argument.
499
500     ;as a result, let's move it into n1 and go:
501     move    #status_str_table,r1 ;our base table for lookup is
502                                     ; status_str_table
503     move    a1,n1                 ;move our status into n1 for indexing
504     move    p:(r1+n1),r0          ;and then we get the address to pass
505                                     ; to our string-display func
506
507     ;we now display the text
508     jsr     display_write_string
509
510     ;end of body code, now restore all registers
511     POP_ACC b
512

```

```

513         move    ssh,n1
514         move    ssl,r1
515         move    ssh,r0
516
517         ;standard C epilogue
518         move    #0+1,n6
519         move    (r6)-n6    ; deallocate local stack space, set ccr flags.
520         tst     a
521         move    x:(r6),ssh ; get the return address.
522         rts     ;return from subroutine
523 ;end Fdisplay_status
524
525
526 ; Fdisplay_title
527 ; Functional Specification
528 ;
529 ; Description:      This function displays the title on the LCD display.
530 ;
531 ; Operation:        Calls display_set_loc to set the display location,
532 ; then clears the location, and then proceeds to write the string to the
533 ; location.
534 ;
535 ; Arguments:
536 ;
537 ; Return Values:    None.
538 ;
539 ; Global Variables: None.
540 ;
541 ; Shared Variables: None.
542 ;
543 ; Local Variables:  None.
544 ;
545 ; Inputs:           None.
546 ;
547 ; Outputs:          None.
548 ;
549 ; Error Handling:   None.
550 ;
551 ; Algorithm:        None.
552 ;
553 ; Data Structures:  None.
554 ;
555 ; Registers changed: None.
556 ;
557 ; Limitations:      None known.
558 ;
559 ; Known Bugs:       None.
560 ;
561 ; Special Notes:
562 ;
563 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
564 ;
565         global  Fdisplay_title
566 Fdisplay_title:                ;C entry point
567
568         ;standard C prologue
569         move    #0,n6          ; k is the amount of local space needed.

```



```

570         move ssh,x:(r6)+ ; save the return address.
571         move (r6)+n6      ; allocate local stack space of size k.
572
573         ;save all registers we use so we can restore later
574         move    r0,ssh
575         move    r1,ssl
576
577         ;we also change b, so preserve it
578         PUSH_ACC a
579
580
581         ;we first clear the title field in the 2nd row
582         move    a1,r1      ;keep our string pointer around
583         move    #>$40,a    ;we like to be in the 2nd row.
584         jsr     display_set_loc
585
586
587         ;and then clear:
588         move    #clear_title_str,r0
589         jsr     display_write_string
590
591
592         ;now write our real title
593         ;(go back to the initial position)
594         jsr     display_set_loc
595
596         ;our string to write has to be in x-space, since the title is volatile
597         move    r1,r0      ;move our string pointer back in place
598         jsr     display_write_string
599
600
601         ;end of body code, now restore all registers
602         POP_ACC a
603
604         move    ssl,r1
605         move    ssh,r0
606
607         ;standard C epilogue
608         move    #0+1,n6
609         move    (r6)-n6    ; deallocate local stack space, set ccr flags.
610         tst     a
611         move    x:(r6),ssh ; get the return address.
612         rts     ;return from subroutine
613 ;end Fdisplay_title
614
615 ; Fdisplay_artist
616 ; Functional Specification
617 ;
618 ; Description:      This function displays the artist on the LCD display.
619 ;
620 ; Operation:        Calls display_set_loc to set the display location,
621 ; then clears the location, and then proceeds to write the string to the
622 ; location.
623 ;
624 ; Arguments:        a1 - pointer to ASCII NULL-term'd string to write
625 ;
626 ; Return Values:    None.

```

```

627 ;
628 ; Global Variables: None.
629 ;
630 ; Shared Variables: None.
631 ;
632 ; Local Variables: None.
633 ;
634 ; Inputs:          None.
635 ;
636 ; Outputs:         Artist name on the display.
637 ;
638 ; Error Handling:   None.
639 ;
640 ; Algorithm:        None.
641 ;
642 ; Data Structures: None.
643 ;
644 ; Registers changed: None.
645 ;
646 ; Limitations:      None known.
647 ;
648 ; Known Bugs:       None.
649 ;
650 ; Special Notes:
651 ;
652 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
653 ;
654     global Fdisplay_artist
655 Fdisplay_artist:                ;C entry point
656
657     ;standard C prologue
658     move #0,n6                ; k is the amount of local space needed.
659     move ssh,x:(r6)+          ; save the return address.
660     move (r6)+n6              ; allocate local stack space of size k.
661
662     ;save all registers we use so we can restore later
663     move r0,ssh
664     move r1,ssl
665
666     ;we also change b, so preserve it
667     PUSH_ACC a
668
669     ;we write to the top left of the display:
670
671     move a1,r1                ;keep our string pointer around
672     move #>$0,a              ;we like to be in the top left corner.
673     jsr display_set_loc
674
675     move #clear_artist_str,r0 ;now we clear
676     jsr display_write_string
677
678     jsr display_set_loc        ;go back to top left corner
679
680     ;our string to write has to be in x-space, since the artist is volatile
681     move r1,r0                ;write our real artist
682     jsr display_write_string
683

```

```

684
685     ;end of body code, now restore all registers
686     POP_ACC a
687
688     move    ssl,r1
689     move    ssh,r0
690
691     ;standard C epilogue
692     move    #0+1,n6
693     move    (r6)-n6    ; deallocate local stack space, set ccr flags.
694     tst     a
695     move    x:(r6),ssh ; get the return address.
696     rts     ;return from subroutine
697 ;end Fdisplay_artist
698
699
700
701 ; Fdisplay_time
702 ; Functional Specification
703 ;
704 ; Description:      This function displays the time on the LCD display.
705 ;
706 ; Operation:        This function divides the time given by 600 to get
707 ; minutes, then calls dec2string, and displays it at the appropriate
708 ; spot. This step is repeated for the next 2 subdivisions: seconds and
709 ; tenths of seconds, each division taking the remainder of the last.
710 ;
711 ; Arguments:        a1 - integer of 10ths of seconds in the song.
712 ;
713 ; Return Values:    None.
714 ;
715 ; Global Variables: None.
716 ;
717 ; Shared Variables: convert_temp_buf - temporary buffer for conversion use
718 ;
719 ; Local Variables:  None.
720 ;
721 ; Inputs:           None.
722 ;
723 ; Outputs:          Time on the screen.
724 ;
725 ; Error Handling:    None.
726 ;
727 ; Algorithm:         None.
728 ;
729 ; Data Structures:  None.
730 ;
731 ; Registers changed: None.
732 ;
733 ; Limitations:       None known.
734 ;
735 ; Known Bugs:        None.
736 ;
737 ; Special Notes:
738 ;
739 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
740 ;

```

```

741     global  Fdisplay_time
742 Fdisplay_time:                ;C entry point
743
744     nop
745     nop
746     rts
747
748     ;standard C prologue
749     move #0,n6                ; k is the amount of local space needed.
750     move ssh,x:(r6)+          ; save the return address.
751     move (r6)+n6              ; allocate local stack space of size k.
752
753     ;save all registers we use so we can restore later
754     PUSH_ACC a
755     PUSH_ACC b
756     move r0,ssh
757     move x0,ssl
758
759
760     move    a,b                ;keep our real time around
761     move    #>$68-9,a          ;we like to be in the bottom right corner.
762                                     ;(we can write $28 characters to a line,
763                                     ; so subtract nine to be right-justified)
764     ;we clear out the time to get rid of prev. digits
765     jsr    display_set_loc
766     move    #clear_time_str,r0
767     jsr    display_write_string
768
769     ;we compare if the time is equal to TIME_NOTIME
770     move    b,a
771
772     cmp     #>TIME_NONE,a
773     jeq     display_time_done    ;if it is equal, do not display anything
774
775     ;now that we're cleared, let's write the minutes:
776     ; (we rely on the builtin C functions to make our life easier)
777     move    #600,x0              ;we divide by 600 tenths of a second ==
778                                     ; one minute
779     jsr     idiv_x0a              ; a = floor(a/x0)
780
781     ;now we convert our minutes to seconds and display:
782     move    #convert_temp_buf,r0
783     jsr     dec2string              ;convert a to a string
784
785     move    #>$68-9,a          ;we like to be in the bottom right corner.
786                                     ;(we can write $28 characters to a line,
787                                     ; so subtract nine to be right-justified)
788     jsr     display_set_loc
789     jsr     display_write_string    ;write minute string
790
791     ;write the separator between min/sec:
792     move    #time_minute_sep,r0
793     jsr     display_write_string
794
795     ;we now get the number of seconds:
796     ; seconds = (tenths of a second % 600) / 10
797     move    b,a                ;retrieve our whole value

```

```

798         move    #>600,x0
799         jsr      imod_x0a ; a = a%600
800
801         ;we divide by ten to get seconds:
802         move     #>10,x0
803         jsr      idiv_x0a ; a = floor(a/10)
804
805         ;and we turn this into a string and blit it:
806         move     #convert_temp_buf,r0
807         jsr      dec2string
808         jsr      display_write_string
809
810         ;we write the divider between tenths and seconds:
811         move     #time_tenths_sep,r0
812         jsr      display_write_string
813
814         ;we now have to deal with tenths, which is easy:
815         ; tenths = tenths % 10
816         move     b,a      ;grab our original value again
817         move     #>10,x0 ;mod 10
818         jsr      imod_x0a ; a = a%10
819
820         ;turn this into a string and blit it to the display:
821         move     #convert_temp_buf,r0
822         jsr      dec2string
823         jsr      display_write_string
824
825         ;and that's it, for displaying the time!
826
827
828
829
830 display_time_done:
831
832         ;end of body code, now restore all registers
833         move     ssl,x0
834         move     ssh,r0
835         POP_ACC b
836         POP_ACC a
837
838         ;standard C epilogue
839         move     #0+1,n6
840         move     (r6)-n6    ; deallocate local stack space, set ccr flags.
841         tst      a
842         move     x:(r6),ssh ; get the return address.
843         rts      ;return from subroutine
844 ;end Fdisplay_time
845
846 ; Fdisplay_volume
847 ; Functional Specification
848 ;
849 ; Description:      This function displays the current volume on the screen.
850 ;
851 ; Operation:        It calls get_atten to display the current volume, then
852 ; shifts this right 3 times (divide by 8) to get the number of dB attenuated.
853 ; This is passed to dec2string, which we write to the display, followed by
854 ; "dB".

```

```

855 ;
856 ; Arguments:      None.
857 ;
858 ; Return Values:  None.
859 ;
860 ; Global Variables: None.
861 ;
862 ; Shared Variables: convert_temp_buf - temporary buffer for binary->ASCII
863 ;                               conversion
864 ;
865 ; Local Variables: None.
866 ;
867 ; Inputs:          None.
868 ;
869 ; Outputs:          Volume on the screen, in units of dB.
870 ;
871 ; Error Handling:   None.
872 ;
873 ; Algorithm:        None.
874 ;
875 ; Data Structures: None.
876 ;
877 ; Registers changed: None.
878 ;
879 ; Limitations:      None known.
880 ;
881 ; Known Bugs:       None.
882 ;
883 ; Special Notes:
884 ;
885 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
886 ;
887     global  Fdisplay_volume
888 Fdisplay_volume                ;C entry point
889
890     ;standard C prologue
891     move #0,n6                ; k is the amount of local space needed.
892     move ssh,x:(r6)+          ; save the return address.
893     move (r6)+n6              ; allocate local stack space of size k.
894
895     ;save all registers we use so we can restore later
896     PUSH_ACC a
897     move r0,ssh
898
899
900     ;we first clear out our portion of the display
901
902     move #>ATTEN_ADDR,a ;we like to be located after the status
903     jsr display_set_loc
904     move #clear_vol_str,r0
905     jsr display_write_string
906
907     ;now, write our "-" sign
908     jsr display_set_loc
909     move #atten_prefix,r0
910     jsr display_write_string
911

```

```
912     jsr  get_atten    ;we now work with the attenuation
913     asr  #ATTEN_SHIFT,a,a ;shift over to get integer dB
914
915     ;and we convert to a number and write it to the screen
916     move  #convert_temp_buf,r0
917     jsr   dec2string
918     jsr   display_write_string
919
920     ;this is followed by writing the " dB"
921
922     move  #atten_suffix,r0
923     jsr   display_write_string
924
925
926
927     ;end of body code, now restore all registers
928     move  ssh,r0
929     POP_ACC a
930
931     ;standard C epilogue
932     move  #0+1,n6
933     move  (r6)-n6      ; deallocate local stack space, set ccr flags.
934     tst   a
935     move  x:(r6),ssh ; get the return address.
936     rts   ;return from subroutine
937 ;end Fdisplay_volume
938
939
940     endsec              ;end display
941
```

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.


```
1 ; keyfunc.inc
2 ;
3 ; This file contains several important definitions that the keypad code
4 ; depends on, such as the definitions for the keycodes.
5 ;
6 ; Revision History
7 ;
8 ; 2011/2/16      Raymond Jimenez      initial revision
9 ; 2011/6/12      Raymond Jimenez      cleaned up for documentation
10 ;
11
12
13 ;if our translated key value is equal to this, don't
14 ; do anything
15 KEY_NOP          EQU      $ffffff
16
17 ;we steal the definitions below from interfac.h in order
18 ; to use them in our assembly:
19
20 KEY_TRACKUP      EQU      0
21 KEY_TRACKDOWN    EQU      1
22 KEY_PLAY         EQU      2
23 KEY_RPTPLAY      EQU      3
24 KEY_FASTFWD      EQU      4
25 KEY_REVERSE      EQU      5
26 KEY_STOP         EQU      6
27 KEY_VOLUP        EQU      7
28 KEY_VOLDOWN      EQU      8
29 KEY_ILLEGAL      EQU      9
30
31 HAVE_KEY         EQU      1
32 NO_KEY           EQU      0
33
34 KEYPAD_ADDR      EQU      $101000
35
36 KEY_VALID_MASK   EQU      $00000f
37
38
```

```

1  ; keyfunc.asm
2  ;
3  ; This file contains procedures related to the keypad, such as getkey and the
4  ; keypad interrupt handler.
5  ;
6  ; keypad_handler - the event handler called when a new key has been pressed
7  ; Fkey_init      - initialization code for the keypad; call before doing
8  ;                  anything related to the keypad (including installing event
9  ;                  handlers)
10 ; Fkey_available - call to see if there is a new key available for processing
11 ; Fgetkey        - call to get a key; blocks on key input
12 ;
13 ; Revision History
14 ;
15 ; 2011/2/16      Raymond Jimenez      initial revision
16 ; 2011/6/12      Raymond Jimenez      cleaned up for documentation
17 ;
18
19
20
21      section keyfunc                ;here's our declaration for our file
22  INCLUDE 'keyfunc.inc'
23  INCLUDE 'macros.inc'
24
25      ;we declare volatile data structures first
26      org x:                        ;keep all the volatile variables in sram/dram
27
28      ; VOLATILE DEFINITONS START HERE
29
30 raw_key:
31      ds          $1                ;we have a word to contain the raw keycode
32
33 raw_havekey:
34      ds          $1                ;and one word as a boolean to see if we have a new key
35
36      ; CONSTANT DEFINITIONS START HERE
37      org p:                        ;keep all constants in P-space for burning
38
39      ;we now define a translation table for our keypad:
40      ;
41      ;our keypad looks sort of like this:
42      ;
43      ; +---+---+---+---+
44      ; | 0 | 1 | 2 | 3 |
45      ; +---+---+---+---+
46      ; | 4 | 5 | 6 |   |
47      ; +---+---+---+---+
48      ; | 8 | 9 | 10 | 7 |
49      ; +---+---+---+---+
50      ; | 12 | 13 | 14 |   |
51      ; +---+---+---+ 15 +
52      ; |   | 11 |   |   |
53      ; +---+---+---+---+
54      ;
55      ; where we have 15 keys in a not-quite-grid-like fashion.
56      ; (numbers are values returned when reading from the keypad)
57

```

```

58
59 key_trans_table:
60     dc     KEY_NOP           ;key 0 - don't do anything
61     dc     KEY_NOP           ;key 1 - don't do anything
62     dc     KEY_NOP           ;key 2 - don't do anything
63     dc     KEY_NOP           ;key 3 - don't do anything
64     dc     KEY_NOP           ;key 4 - don't do anything
65     dc     KEY_TRACKUP       ;key 5 - track up
66     dc     KEY_VOLUP         ;key 6 - increase volume
67     dc     KEY_RPTPLAY       ;key 7 - repeat play (loop)
68     dc     KEY_REVERSE       ;key 8 - reverse
69     dc     KEY_STOP          ;key 9 - stop
70     dc     KEY_FASTFWD       ;key 10 - fast forward
71     dc     KEY_NOP           ;key 11 - don't do anything
72     dc     KEY_NOP           ;key 12 - don't do anything
73     dc     KEY_TRACKDOWN     ;key 13 - track down
74     dc     KEY_VOLDOWN       ;key 14 - decrease volume
75     dc     KEY_PLAY          ;key 15 - play
76     ;we should never get higher values of keys, otherwise
77     ; something is wrong.
78     dc     KEY_NOP           ;key 16 - don't do anything, but it's
79                                ; odd that it's even here...
80     dc     KEY_NOP           ;key 17 - don't do anything
81     dc     KEY_NOP           ;key 18 - don't do anything
82     dc     KEY_NOP           ;key 19 - don't do anything
83     dc     KEY_NOP           ;key 20 - don't do anything
84
85
86     ; PROGRAM CODE STARTS HERE
87
88     ;and finally, program code
89
90 ; keypad_handler
91 ; Functional Specification
92 ;
93 ; Description:      keypad_handler is the interrupt handler for the keypad.
94 ; It sets up raw_key and raw_havekey so that the other key functions work.
95 ;
96 ; Operation:        Executes a move instruction to move the raw
97 ; keycode into the variable, and then moves HAVEKEY into raw_havekey
98 ;
99 ; Arguments:         None.
100 ;
101 ; Return Values:     None.
102 ;
103 ; Global Variables:  None.
104 ;
105 ; Shared Variables:  None.
106 ;
107 ; Local Variables:  None.
108 ;
109 ; Inputs:            None.
110 ;
111 ; Outputs:           None.
112 ;
113 ; Error Handling:    None.
114 ;

```

```

115 ; Algorithm:      None.
116 ;
117 ; Data Structures: None.
118 ;
119 ; Registers changed: None.
120 ;
121 ; Limitations:     None known.
122 ;
123 ; Known Bugs:      None.
124 ;
125 ; Special Notes:
126 ;
127 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
128 ;
129     global keypad_handler
130 keypad_handler:                ;assembly entry point
131     ;remember to save all registers changed! interrupt handler, so very
132     ; important!
133     move    x0,ssh
134
135     move    x:KEYPAD_ADDR,x0    ;move our key into a temp buffer
136                                   ; (this also clears the interrupt by
137                                   ; reading from the keypad chip)
138     move    x0,x:raw_key        ; then into our variable
139
140     move    #HAVE_KEY,x0        ;and then signal we have a key
141     move    x0,x:raw_havekey
142
143     ;restore registers
144     move    ssh,x0
145
146     rti        ;interrupt handlers return w/ rti
147 ;end keypad_handler
148
149 ; Fkey_init
150 ; Functional Specification
151 ;
152 ; Description:      Fkey_init should be called before any other key functions
153 ; are called in order to initialize the data structures.
154 ;
155 ; Operation:        Fkey_init clears out raw_havekey to zero.
156 ;
157 ; Arguments:        None.
158 ;
159 ; Return Values:    None.
160 ;
161 ; Global Variables: None.
162 ;
163 ; Shared Variables: None.
164 ;
165 ; Local Variables: None.
166 ;
167 ; Inputs:           None.
168 ;
169 ; Outputs:          None.
170 ;
171 ; Error Handling:    None.

```

```

172 ;
173 ; Algorithm:      None.
174 ;
175 ; Data Structures: None.
176 ;
177 ; Registers changed: a.
178 ;
179 ; Limitations:    None known.
180 ;
181 ; Known Bugs:     None.
182 ;
183 ; Special Notes:
184 ;
185 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
186 ;
187     global  Fkey_init
188 Fkey_init:                ;C entry point
189
190     ;standard C prologue
191     move #0,n6            ; k is the amount of local space needed.
192     move ssh,x:(r6)+      ; save the return address.
193     move (r6)+n6         ; allocate local stack space of size k.
194
195     ;save all of the registers we use
196     move    r0,ssh        ;save r0
197
198     move    #NO_KEY,r0
199     move    r0,x:raw_havekey
200
201     ;restore registers
202     move    ssh,r0
203
204     ;standard C epilogue
205     move #0+1,n6
206     move (r6)-n6         ; deallocate local stack space, set ccr flags.
207     tst    a
208     move x:(r6),ssh      ; get the return address.
209     ;save all registers we use so we can restore later
210
211     rts            ;return from subroutine
212 ;end Fkey_init
213
214 ; Fkey_available
215 ; Functional Specification
216 ;
217 ; Description:      Fkey_available returns 0 if there is no key, otherwise
218 ; non-zero.
219 ;
220 ; Operation:        Fkey_available checks if raw_havekey has been set. If so,
221 ; it checks the value of raw_key to see if it is a valid key; if not, it clears
222 ; raw_havekey and returns. If it is a valid key, it returns non-zero.
223 ;
224 ; Arguments:        None.
225 ;
226 ; Return Values:    a1 - non-zero if there is a valid key waiting
227 ;
228 ; Global Variables: None.

```

```

229 ;
230 ; Shared Variables: None.
231 ;
232 ; Local Variables: None.
233 ;
234 ; Inputs:           None.
235 ;
236 ; Outputs:          None.
237 ;
238 ; Error Handling:   None.
239 ;
240 ; Algorithm:        None.
241 ;
242 ; Data Structures:  None.
243 ;
244 ; Registers changed: a.
245 ;
246 ; Limitations:      None known.
247 ;
248 ; Known Bugs:       None.
249 ;
250 ; Special Notes:
251 ;
252 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
253 ;
254     global Fkey_available
255 Fkey_available:                ;C entry point
256
257     ;standard C prologue
258     move #0,n6                ; k is the amount of local space needed.
259     move ssh,x:(r6)+          ; save the return address.
260     move (r6)+n6              ; allocate local stack space of size k.
261
262     ;save all of the registers we use
263     move r0,ssh               ;save both r0/n0
264     move n0,ssl
265
266     move x:raw_havekey,a      ;we check if we have a key
267     tst a                     ;do we have a key?
268     jeq key_avail_done       ;if not, we're done
269
270
271     move #key_trans_table,r0  ;start the lookup
272     move x:raw_key,a          ;move our key into a so we can clear
273     and #>KEY_VALID_MASK,a    ;clear extreaneous high bits
274     move a1,n0                ;we use the key as an index
275
276     move p:(r0+n0),a          ;grab the keycode into a
277     eor #KEY_NOP,a           ;XOR it with the NOP command
278                               ; -> will return zero only if NOP
279                               ; otherwise, guaranteed to be non-zero
280
281 key_avail_done:
282
283     ;end of body code, now restore all registers
284     move ssl,n0
285     move ssh,r0

```

```

286
287     ;standard C epilogue
288     move #0+1,n6
289     move (r6)-n6    ; deallocate local stack space, set ccr flags.
290     tst  a
291     move x:(r6),ssh ; get the return address.
292     ;save all registers we use so we can restore later
293
294     rts      ;return from subroutine
295 ;end function
296
297
298 ; Fgetkey
299 ; Functional Specification
300 ;
301 ; Description:      Fgetkey returns the translated keycode of a key; it blocks
302 ;if there is no key available.
303 ;
304 ; Operation:        Fgetkey calls Fkey_available to see if there is a key ready,
305 ; if not it loops until there is. It then grabs the value of raw_key, clears
306 ; unnecessary bits, and looks up that value in a table, which is then put into
307 ; a1.
308 ;
309 ; Arguments:        None.
310 ;
311 ; Return Values:    a1 - translated keycode
312 ;
313 ; Global Variables: None.
314 ;
315 ; Shared Variables: None.
316 ;
317 ; Local Variables:  None.
318 ;
319 ; Inputs:           None.
320 ;
321 ; Outputs:          None.
322 ;
323 ; Error Handling:    None.
324 ;
325 ; Algorithm:         None.
326 ;
327 ; Data Structures:  None.
328 ;
329 ; Registers changed: a.
330 ;
331 ; Limitations:       None known.
332 ;
333 ; Known Bugs:        None.
334 ;
335 ; Special Notes:
336 ;
337 ; Revision History:  2011/02/16    Raymond Jimenez    first revision
338 ;
339     global  Fgetkey
340 Fgetkey:      ;C entry point
341

```

```

342         ;standard C prologue
343         move #0,n6          ; k is the amount of local space needed.
344         move ssh,x:(r6)+    ; save the return address.
345         move (r6)+n6        ; allocate local stack space of size k.
346
347         ;save all of the registers we use
348         move r0,ssh          ;save both r0/n0
349         move n0,ssl
350
351     getkey_block_loop:
352         jsr    Fkey_available    ;check if there are keys available
353         jeq    getkey_block_loop ;if not, keep on looping
354
355         move   #key_trans_table,r0 ;put the lookup table in a base pointer
356         move   x:raw_key,a          ;move our key into a so we can clear
357         and    #>KEY_VALID_MASK,a  ;clear extreaneous high bits
358         move   a1,n0                ;we use the key as an index
359
360         move   p:(r0+n0),a          ;grab the keycode into a, we're done
361
362         move   #NO_KEY,r0
363         move   r0,x:raw_havekey     ;now we no longer have a key
364
365         ;end of body code, now restore all registers
366         move   ssl,n0
367         move   ssh,r0
368
369         ;standard C epilogue
370         move #0+1,n6
371         move (r6)-n6          ; deallocate local stack space, set ccr flags.
372         tst    a
373         move x:(r6),ssh      ; get the return address.
374         ;save all registers we use so we can restore later
375
376         rts          ;return from subroutine
377 ;end Fgetkey
378
379
380
381
382
383         endsec          ;end keypad
384

```


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.

```

1
2 ;the various IDE register addresses
3 ; some are read only, others write only,
4 ; most r/w. please check the ATA spec for details.
5
6 ;the addresses come from how we've wired up the
7 ; bus to the IDE connector.
8
9
10 IDE_DATA_REG          EQU      $103000
11 IDE_FEATURE_REG       EQU      $103100
12 IDE_ERROR_REG         EQU      $103100
13 IDE_SEC_CT_REG        EQU      $103200
14 IDE_SEC_NUM_REG       EQU      $103300
15 IDE_CYL_LOW_REG       EQU      $103400
16 IDE_CYL_HIGH_REG      EQU      $103500
17 IDE_DEV_HEAD_REG      EQU      $103600
18 IDE_STATUS_REG        EQU      $103700
19 IDE_COMMAND_REG       EQU      $103700
20
21 IDE_ALT_STATUS_REG     EQU      $102e00
22 IDE_ALT_CONTROL_REG   EQU      $102e00
23
24 ;all registers are 8 bits long
25 IDE_REG_LENGTH        EQU      8
26
27
28 ;now we define various commands
29
30 ;to enable LBA and device 0, write to dev/head reg:
31 IDE_SELECT_0_LBA      EQU      $40
32
33 ;to read from the device:
34 IDE_READ_SEC          EQU      $20
35
36 ;to identify:
37 IDE_IDENTIFY          EQU      $ec
38
39 ;to set faster speeds
40 IDE_SET_FEATURES      EQU      $ef
41
42 IDE_SET_TRANSFER_MODE  EQU      $03
43 IDE_SET_FASTER_PIO    EQU      $0a ;support PIO-2, no need for IORDY
44
45 ;and now, various bits within the status buffer:
46 IDE_STAT_DRDY         EQU      6
47 IDE_STAT_BSY          EQU      7
48 IDE_STAT_DRQ          EQU      3
49 IDE_STAT_ERR          EQU      0
50
51 ;an IDE block is always 256 words:
52 IDE_BLOCK_SIZE        EQU      256
53
54 IDE_VALID_DATA         EQU      $ffff
55
56

```

```
1
2 ; ide.asm
3 ;
4 ; This file offers several IDE routines, such as setup and other things.
5 ;
6
7     section ide
8     INCLUDE 'macros.inc'
9     INCLUDE 'ide.inc'
10    org     x:
11
12 ;no variables yet, i think
13
14    org     p:                ;we want to be located in program space
15
16 ; Fsetup_ide
17 ; Functional Specification
18 ;
19 ; Description:      Fsetup_ide sets up the IDE drive by selecting device 0 and
20 ; setting it to LBA mode. We assume that a drive is attached and in the master
21 ; position.
22 ;
23 ; Operation:        It writes IDE_SELECT_0_LBA to address IDE_DEVHEAD_REG, which
24 ; is the device/head register. It then loops until the drive is ready for
25 ; activity that is, until DRDY is set.
26 ;
27 ; Arguments:        None.
28 ;
29 ; Return Values:     None.
30 ;
31 ; Global Variables: None.
32 ;
33 ; Shared Variables: None.
34 ;
35 ; Local Variables:  None.
36 ;
37 ; Inputs:            None.
38 ;
39 ; Outputs:           None.
40 ;
41 ; Error Handling:    None.
42 ;
43 ; Algorithm:         None.
44 ;
45 ; Data Structures:  None.
46 ;
47 ; Registers changed: None.
48 ;
49 ; Limitations:       None known.
50 ;
51 ; Known Bugs:        None.
52 ;
53 ; Special Notes:
54 ;
55 ; Revision History: 2011/03/08    Raymond Jimenez    first revision
56 ;
```

```

57     global  Fsetup_ide
58 Fsetup_ide:      ;C entry point
59     ;standard C prologue
60     move     #0,n6          ;k is the amount of local space
61     move     ssh,x:(r6)+    ;save the return address
62     move     (r6)+n6        ;allocate local stack space of size k
63
64     ;save all registers we use so we can restore later
65     move     r0,ssh
66
67     ;wait until the drive is ready for selecting
68     jsr      wait_for_ide
69
70     ;select the device
71     move     #>IDE_SELECT_0_LBA,r0
72     move     r0,x:IDE_DEV_HEAD_REG
73
74     move     #>IDE_IDENTIFY,r0
75     move     r0,x:IDE_COMMAND_REG
76
77     ;wait until it has reported back
78     jsr      wait_for_ide
79
80     ;we now set a faster transfer mode
81     move     #>IDE_SET_TRANSFER_MODE,r0
82     move     r0,x:IDE_FEATURE_REG
83
84     move     #>IDE_SET_FASTER_PIO,r0
85     move     r0,x:IDE_SEC_CT_REG
86
87     move     #>IDE_SET_FEATURES,r0
88     move     r0,x:IDE_COMMAND_REG
89
90     ;wait until it has reported back
91     jsr      wait_for_ide
92
93     ;restore registers
94     move     ssh,r0
95
96     ;standard C epilogue
97     move     #0+1,n6
98     move     (r6)-n6        ;deallocate local stack space, set ccr flags
99     tst      a
100    move     x:(r6),ssh      ;get return address
101    nop
102    rts      ;return from subroutine
103 ;end Fsetup_ide
104
105
106 ; wait_for_ide
107 ; Functional Specification
108 ;
109 ; Description:      wait_for_ide simply reads the status register and waits
110 ; until BSY and DRDY are cleared and set, respectively.
111 ;
112 ; Operation:      reads from IDE_STATUS_REG, checks BSY and DRDY bits,
113 ; loops if not ready

```

```

114 ;
115 ; Arguments:      None
116 ;
117 ; Return Values:  None.
118 ;
119 ; Global Variables: None.
120 ;
121 ; Shared Variables: None.
122 ;
123 ; Local Variables: None.
124 ;
125 ; Inputs:         None.
126 ;
127 ; Outputs:        None.
128 ;
129 ; Error Handling:  None.
130 ;
131 ; Algorithm:       None.
132 ;
133 ; Data Structures: None.
134 ;
135 ; Registers changed: None.
136 ;
137 ; Limitations:     None known.
138 ;
139 ; Known Bugs:      None.
140 ;
141 ; Special Notes:
142 ;
143 ; Revision History: 2011/03/08   Raymond Jimenez   first revision
144 ;
145     global  wait_for_ide
146 wait_for_ide:                ;assembly entry point
147
148     ;save all registers we use so we can restore later
149     move    r0,ssh
150
151 wait_for_ide_loop:
152     move    x:IDE_STATUS_REG,r0
153     nop     ;GDB delay
154     jset    #IDE_STAT_BSY,r0,wait_for_ide_loop
155
156     ;end of body code, now restore all registers
157     move    ssh,r0
158     nop
159     rts     ;return from subroutine
160 ;end wait_for_ide
161
162
163 ; Fget_blocks
164 ; Functional Specification
165 ;
166 ; Description:      Fget_blocks grabs a number of blocks from the specified
167 ; address on the IDE disk.
168 ;
169 ; Operation:        It first sets up the head registers, and then commits the
170 ; command to read from the disk. We don't actually detect errors; we depend on

```

```

171 ; the IDE drive to stop giving us data, resulting in returning less blocks
172 ; than the calling function expects.
173 ;
174 ; Arguments:      first (a) - block number to start reading from
175 ;                  second (b) - number of blocks to read
176 ;                  third (r6-1) - (on stack) - pointer of address to write to
177 ;
178 ; Return Values:  a - number of blocks read
179 ;
180 ; Global Variables: None.
181 ;
182 ; Shared Variables: None.
183 ;
184 ; Local Variables: None.
185 ;
186 ; Inputs:         None.
187 ;
188 ; Outputs:        None.
189 ;
190 ; Error Handling:  None.
191 ;
192 ; Algorithm:       None.
193 ;
194 ; Data Structures: None.
195 ;
196 ; Registers changed: a
197 ;
198 ; Limitations:     None known.
199 ;
200 ; Known Bugs:      None.
201 ;
202 ; Special Notes:
203 ;
204 ; Revision History: 2011/03/08    Raymond Jimenez    first revision
205 ;
206     global  Fget_blocks
207 Fget_blocks:      ;C entry point
208     ;standard C prologue
209     move    #0,n6          ;k is the amount of local space
210     move    ssh,x:(r6)+    ;save the return address
211     move    (r6)+n6        ;allocate local stack space of size k
212
213     ;save all registers we use so we can restore later
214     move    r0,ssh
215     move    r1,ssl
216     PUSH_ACC b
217
218     move    x:(r6-2),r1    ;grab our pointer (arg 3) off the stack
219
220     ;note that a carries an unsigned long int, which is 2 words.
221     ; as a result, we have MSW in a1, LSW in a0.
222
223     ;setup the registers for the read
224     move    a0,x:IDE_SEC_NUM_REG
225     move    b1,x:IDE_SEC_CT_REG
226
227     ;since we only have a 24-bit counter for blocks, assume [15:8] will

```

```

228      ; go to IDE_CYL_LOW
229      asr      #IDE_REG_LENGTH,a,a
230      move     a0,x:IDE_CYL_LOW_REG
231
232      ;and then [23:16] go to IDE_CYL_HIGH
233      asr      #IDE_REG_LENGTH,a,a
234      move     a0,x:IDE_CYL_HIGH_REG
235
236      ;and then we need to put [27:24] in the last 4 bits of
237      ; IDE_DEV_HEAD_REG
238      asr      #IDE_REG_LENGTH,a,a
239      move     a0,a1
240      and      #>$ff,a
241      or       #>IDE_SELECT_0_LBA,a
242      move     a1,x:IDE_DEV_HEAD_REG
243
244      ;now execute the command
245      move     #>IDE_READ_SEC,r0
246      move     r0,x:IDE_COMMAND_REG
247
248      ;wait until it has reported back
249      jsr      wait_for_ide
250
251
252      ;we now clear a and use it as our data word counter
253      clr      a
254
255      ;we can get a maximum of b1 words, so we start looping and comparing
256 get_all_blocks:
257      nop
258
259      ;we can get IDE_BLOCK_SIZE words at least, so limit it to that
260      ; we are guaranteed at least one block per ATA spec.
261      do #IDE_BLOCK_SIZE,get_block_loop_end
262 get_block_loop:
263      nop
264
265      move     x:IDE_DATA_REG,b
266      and      #>IDE_VALID_DATA,b
267      move     b1,x:(r1)+
268
269      jsr      wait_for_ide
270
271      ;spacing nop
272      nop
273      nop
274      nop
275      nop
276      nop
277      nop
278 get_block_loop_end:
279
280
281      add      #>1,a
282
283      ;and then see if we have more data to run:
284      move     x:IDE_STATUS_REG,b      ;check the status reg

```

```
285         and    #>(1<<IDE_STAT_DRQ),b    ;check the DRQ bit
286         jeq     get_all_blocks_end        ;if equal, get out of here, no more
287                                         ; blocks to read
288         ;spacing nop
289         nop
290         nop
291         nop
292         jmp     get_all_blocks
293
294 get_all_blocks_end:
295
296
297         ;restore registers
298         POP_ACC b
299         move    ssl,r1
300         move    ssh,r0
301
302
303         ;standard C epilogue
304         move    #0+1,n6
305         move    (r6)-n6        ;deallocate local stack space, set ccr flags
306         tst     a
307         move    x:(r6),ssh    ;get return address
308         nop
309         rts     ;return from subroutine
310 ;end Fget_blocks
311
312         endsec
313
314
315
```


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.

```

1
2
3 ; the following are helpful constants when dealing with ESSIO, the
4 ; VLSI VS1053b MP3 decoder chip. data and control are both run over
5 ; ESSIO.
6
7 ; START DECODER SECTION
8
9 ;the ESSIO ports expect left aligned data, so we shift over this number of bits:
10 ESSIO_BYTE_SHIFT EQU      16
11 ;we deal with 16-bit words in SCI mode
12 ESSIO_SCI_SHIFT EQU       8
13
14
15 ;and a mask just to mask off a single byte, for macro convenience
16 BYTE_MASK EQU            $f
17
18
19 ;initial ESSIO_CRA, set to 1.6MHz per DAC startup. once we change clock freq up,
20 ; we can do higher serial too.
21 ; we also start with 16 bit words since we're writing to SCI initially.
22
23 ESSIO_CRA_1ST EQU        $100831      ; 0001 0000 0000 1000 0011 0010
24                                         ; 0--- --0- ---- -000 ---- reserved
25                                         ; -0-- ---- ---- ---- ---- SCI as
26                                         flag
27                                         ; --01 0--- ---- ---- ---- 16b
28                                         words
29                                         ; ---- -0-- ---- ---- ---- align
30                                         left
31                                         ; ---- ---0 0000 ---- ----
32                                         sequential transmit mode
33                                         ; ---- ---- ---- 1--- ---- no
34                                         prescaler
35                                         ; ---- ---- ---- ---- 0011 0010 divide
36                                         by 49+1
37                                         ;
38                                         (80/50 =
39                                         1.6MHz)
40
41 ;for initialization we don't depend on interrupts, instead doing polling.
42
43 ESSIO_CRB_1ST EQU        $001d3c      ; 0000 0000 0001 1101 0011 1100
44                                         ; 0--- ---- ---- ---- ---- no RX
45                                         ex. interrupt
46                                         ; -0-- ---- ---- ---- ---- no TX
47                                         ex. interrupt
48                                         ; --0- ---- ---- ---- ---- no RX
49                                         slot interrupt
50                                         ; ---0 ---- ---- ---- ---- no TX
51                                         slot interrupt
52                                         ; ---- 0--- ---- ---- ---- no RX
53                                         interrupt
54                                         ; ---- -0-- ---- ---- ---- no TX
55                                         interrupt
56                                         ; ---- --0- ---- ---- ---- RX
57                                         enabled (will do later)
58                                         ; ---- ---0 ---- ---- ---- TX0

```

```

43         (STD) enabled (will do later)
44         ; ---- 0--- ---- TX1
         (SC0) disabled
45         ; ---- -0-- ---- TX2
         (SC1) enabled (will do later)
46         ; ---- --0- ---- normal
         mode
47         ; ---- ---1 ----
         synchronous mode
48         ; ---- 1--- ---- clock
         out data on falling edge
49         ; ---- -1-- ---- negative
         edge triggers word (positive is end of
         word)
50         ; ---- --0- ---- frame
         sync w/ first bit
51         ; ---- ---1 0--- ---- bit sync
         for SCI, need to transmit consecutively
52         ; ---- -0-- ---- MSB
         shifted first
53         ; ---- --1- ---- internal
         clock
54         ; ---- ---1 ---- SC2 is
         an output
55         ; ---- 1--- SC1 is
         an output
56         ; ---- -1-- SC0 is
         an output
57         ; ---- --00 don't
         care, don't use flags
58
59 ;normal mode, we run at decoder 55MHz, so since max serial is CLKI/7, we get to
60 ; run serial at 6.66MHz now
61 ESSI0_CRA_SCI EQU (((ESSI0_CRA_1ST&(~M_PM)))|$05)
62 ESSI0_CRB_SCI EQU ESSI0_CRB_1ST
63
64 ;we switch modes between SCI and SDI - for data, we use SC2 as a chip select,
65 ; auto-selecting SDI whenever we move in a byte of data.
66 ;
67 ; (this is the only reason we do SDI - so we can enable DMA and write to our
68 ; decoder quite fast)
69
70
71 ESSI0_CRA_DATA EQU (((ESSI0_CRA_1ST&(~M_PM)))|$05)
72 ESSI0_CRB_DATA EQU $003c3c ; 0000 0000 0011 1100 0011 0000
73 ; 0--- ---- no RX
         ex. interrupt
74 ; -0-- ---- no TX
         ex. interrupt
75 ; --0- ---- no RX
         slot interrupt
76 ; ---0 ---- no TX
         slot interrupt
77 ; ---- 0--- ---- no RX
         interrupt
78 ; ---- -0-- ---- no TX
         interrupt

```

```

79      ; ---- --0- ---- ---- ---- RX
      disabled
80      ; ---- --0- ---- ---- ---- TX0
      (STD) enabled (will do later)
81      ; ---- ---- 0--- ---- ---- TX1
      (SC0) disabled
82      ; ---- ---- -0-- ---- ---- TX2
      (SC1) disabled
83      ; ---- ---- --1- ---- ---- network
      mode (on-demand mode)
84      ; ---- ---- --1- ---- ----
      synchronous mode
85      ; ---- ---- ---- 1--- ---- clock
      out data on falling edge
86      ; ---- ---- ---- -1-- ---- negative
      edge triggers word (positive is end of
      word)
87      ; ---- ---- ---- --0- ---- frame
      sync w/ first bit
88      ; ---- ---- ---- --0 0--- ---- word
      sync for data (XDCS asserted byte-wise)
89      ; ---- ---- ---- -0-- ---- MSB
      shifted first
90      ; ---- ---- ---- --1- ---- internal
      clock
91      ; ---- ---- ---- ---- --1- ---- SC2 is
      an output
92      ; ---- ---- ---- ---- 0--- SC1 is
      an output
93      ; ---- ---- ---- ---- -0-- SC0 is
      an output
94
95
96  ESSI0_PCRC_SCI EQU    $00003a      ; 0000 0000 0000 0000 0011 1010
97                                     ; 0000 0000 0000 0000 00-- ---- reserved
98                                     ; ---- ---- ---- ---- --1- ---- TX
      enabled
99      ; ---- ---- ---- ---- ---- --1- ---- RX
      enabled
100     ; ---- ---- ---- ---- ---- 1--- CLK
      enabled
101     ; ---- ---- ---- ---- ---- -0-- SC2
      disabled
102     ; ---- ---- ---- ---- ---- --1- SC1
      enabled
103     ; ---- ---- ---- ---- ---- ---0 SC0
      disabled
104
105  ESSI0_PCRC_DATA EQU    $00002c      ; 0000 0000 0000 0000 0010 1101
106                                     ; 0000 0000 0000 0000 00-- ---- reserved
107                                     ; ---- ---- ---- ---- --1- ---- TX
      enabled
108     ; ---- ---- ---- ---- ---- ---0 ---- RX
      enabled
109     ; ---- ---- ---- ---- ---- 1--- CLK
      enabled
110     ; ---- ---- ---- ---- ---- -1-- SC2

```

```

110             enabled
111             ; ---- ---- ---- ---- ---- --0- SC1
112             enabled
113             ; ---- ---- ---- ---- ---- ---0 SC0
114             disabled
115
116 ;see errata for mask 4H80G, we find that GPIO pins need to be programmed to
117 output
118 ; anyway. this is probably why our shit's not working OTL.
119
120 ESSIO_PRRC      EQU      $00003f      ; 0000 0000 0000 0000 0010 1111
121                                     ; 0000 0000 0000 0000 00-- ---- reserved
122                                     ; ---- ---- ---- ---- ---- --10 1111 all pins
123                                     output (except for RX)
124 ESSIO_PDRC      EQU      $00003f      ;set all outputs high if disabled
125
126 ;we now define some commands/registers for the VS1053 chip:
127
128 XDSC_HIGH       EQU      ($ffff<<ESSI_SCI_SHIFT)
129 XDSC_LOW        EQU      ($0)
130
131 XCS_HIGH        EQU      ($ffff<<ESSI_SCI_SHIFT)
132 XCS_LOW         EQU      ($0)
133
134 ;please note that all data is shifted over 16 bits as ESSI expects
135 ; left-aligned data.
136
137 ;write/read commands, shifted over another 8 since they're the first 8 bits
138 SCI_WRITE_CMD   EQU      ($02<<ESSI_BYTE_SHIFT)
139 SCI_READ_CMD    EQU      ($03<<ESSI_BYTE_SHIFT)
140
141 ;register addresses
142 SCI_MODE        EQU      ($00<<ESSI_SCI_SHIFT)
143 SCI_CLOCKF      EQU      ($03<<ESSI_SCI_SHIFT)
144 SCI_VOL         EQU      ($0B<<ESSI_SCI_SHIFT)
145 SCI_WRAM        EQU      ($06<<ESSI_SCI_SHIFT)
146 SCI_WRAMADDR    EQU      ($07<<ESSI_SCI_SHIFT)
147 SCI_AIADDR      EQU      ($0a<<ESSI_SCI_SHIFT)
148
149 ;register wait times, measured in 100's of ns
150 SET_CLOCK_WAIT  EQU      977          ;worst case scenario: 1200 XTALI,
151                                     12.288MHz crystal
152
153 ;for other wait times, we assume MP3 decoder has been set to 50MHz for CLKI
154 SET_VOL_WAIT    EQU      16           ;80 CLKI
155 SET_WRAM_WAIT   EQU      20           ;100 CLKI
156 SET_WRAMADDR_WAIT EQU      20         ;100 CLKI
157 SET_MODE_WAIT   EQU      20           ;100 CLKI
158 SET_AIADDR_WAIT EQU      42           ;210 CLKI
159
160 ;we left-align all of our data words to be written
161 SCI_CLOCKF_VALUE EQU      ($c000<<ESSI_SCI_SHIFT)
162                                     ; 1100 0000 0000 0000
163                                     ; 110- ---- ---- ---- 4.5x multiplier,
164                                     max'd out

```

```

161                                     ; ---0 0--- ---- ---- no turbo mode,
162                                     ; ----- -000 0000 0000 12.288 MHz crystal
163                                     ; (default)
164
165 SCI_VOL_VALUE EQU ($0000<<ESSI_SCI_SHIFT)
166                                     ;max volume, DAC does vol control
167
168 SCI_MODE_VALUE EQU ($0800<<ESSI_SCI_SHIFT) ;allow SDI tests
169 SCI_MODE_CANCEL_VALUE EQU ($0808<<ESSI_SCI_SHIFT) ;cancel current playback
170 SCI_AIADDR_VALUE EQU ($4022<<ESSI_SCI_SHIFT) ;sine sweep test
171
172
173 ;MEMORY MAPPED 1053 REGISTERS (accessable via WRAMADDR/WMRAM)
174 ;there a number of 1053-memory-mapped registers that we need to write to:
175 GPIO_DDR_ADDR EQU ($C017<<ESSI_SCI_SHIFT)
176
177 I2S_CONFIG_ADDR EQU ($C040<<ESSI_SCI_SHIFT)
178 ENDFILLBYTE_ADDR EQU ($1e06<<ESSI_SCI_SHIFT)
179
180
181
182 ;and we set these registers to enable I2S per the spec:
183
184 GPIO_DDR_VALUE EQU ($00f0<<ESSI_SCI_SHIFT) ;set GPIO pins to all output
185 I2S_CONFIG_VALUE EQU ($000d<<ESSI_SCI_SHIFT) ;set I2S, MCLK, 96khz
186
187
188 ; the following deals with data transmission via DMA, etc.
189
190 DCR2 EQU $4a5a50 ; 0100 1010 0101 1010 0101 0000
191                                     ; 0--- ---- ---- ---- ---- not
192                                     ; -1-- ---- ---- ---- ----
193                                     ; interrupt enabled
194                                     ; --00 1--- ---- ---- ---- word
195                                     ; transfer, DE-enabled
196                                     ; ---- -01- ---- ---- ---- normal
197                                     ; priority DMA
198                                     ; ---- ---0 ---- ---- ---- not
199                                     ; continuous
200                                     ; ---- ---- 0101 1--- ---- ---- trigger
201                                     ; on ESSI0 TX empty
202
203                                     ; ---- ---- ---- -0-- ---- ---- not 3d
204                                     ; mode
205                                     ; ---- ---- ---- --10 0--- ---- dest
206                                     ; (ESSI) no update
207                                     ; ---- ---- ---- -101 ---- source
208                                     ; (buffer) post-inc
209                                     ; ---- ---- ---- 00--
210                                     ; destination in X (I/O)
211                                     ; ---- ---- ---- ---- --00 source in
212                                     ; X (buffer)

```

```

204 ;constants for our DMA/transfer logic
205 NEEDS_KICKSTART EQU 1
206 NO_KICKSTART EQU 0
207 DREQ_CTL_PRIO EQU %110 ;set to edge-trigger, priority 1, we can transmit up to
208                          ; 32 bytes safely due to safety zone, even after DREQ is
209                          ; lowered
210 NO_BUFFER_LEN EQU 0
211
212
213
214
215 ;the following are helpful constants when dealing with our DAC, the Wolfson
216 ; WM8741. we interface to it via a write-only control interface on ESSI1.
217
218 ; START DAC SECTION
219
220 ESSI1_CRA EQU $100806 ; 0001 0000 0000 1000 0000 0110
221                          ; 0--- --0- ---- -000 ---- reserved
222                          ; -0-- ---- ---- ---- SC1 as
223                          flag
224                          ; --01 0--- ---- ---- 16b
225                          words
226                          ; ---- -0-- ---- ---- align
227                          left
228                          ; ---- ---0 0000 ---- on-demand mode
229                          ; ---- ---- 1--- ---- no
230                          prescaler
231                          ; ---- ---- ---- 0000 0110 divide
232                          by 6+1
233                          ; (80/7 =
234                          11.428)
235
236 ;we use no interrupts with the DAC, because it's fast enough for us just to use
237 ; synchronous changes, with minimal lag.
238
239 ESSI1_CRB EQU $003c30 ; 0000 0000 0001 1100 0011 0000
240                          ; 0--- ---- ---- ---- no RX
241                          ex. interrupt
242                          ; -0-- ---- ---- ---- no TX
243                          ex. interrupt
244                          ; --0- ---- ---- ---- no RX
245                          slot interrupt
246                          ; ---0 ---- ---- ---- no TX
247                          slot interrupt
248                          ; ---- 0--- ---- ---- no RX
249                          interrupt
250                          ; ---- -0-- ---- ---- no TX
251                          interrupt
252                          ; ---- --0- ---- ---- RX
253                          disabled
254                          ; ---- ---0 ---- ---- TX0
255                          (STD) disabled (enabled later)
256                          ; ---- 0--- ---- ---- TX1
257                          (SC0) disabled
258                          ; ---- ---- -0-- ---- TX2
259                          (SC1) disabled

```

```

244             ; ---- -1- ---- - network
                (on-demand) mode
245             ; ---- -1- ---- -
                synchronous mode
246             ; ---- 1- ---- - clock
                out data on falling edge
247             ; ---- -1- ---- - negative
                edge triggers word (positive is end of
                word)
248             ; ---- -0- ---- - frame
                sync w/ first bit
249             ; ---- -0 0- ----
                word-long sync to get positive edge at
                end
250             ; ---- -0- ---- - MSB
                shifted first
251             ; ---- -1- ---- - internal
                clock
252             ; ---- -1- ---- - SC2 is
                an output
253             ; ---- 0- ---- - SC1 is
                GPIO, don't care
254             ; ---- -0- ---- - SC0 is
                GPIO, don't care
255             ; ---- -00 ---- - don't
                care, don't use flags
256
257 ESSI1_PCRD      EQU      $00002c      ; 0000 0000 0000 0000 0010 1100
258                                     ; 0000 0000 0000 0000 00-- ---- reserved
259                                     ; ---- -1- ---- - TX
                enabled
260             ; ---- -0- ---- - RX is
                GPIO
261             ; ---- 1- ---- - CLK
                enabled
262             ; ---- -1- ---- - SC2
                enabled
263             ; ---- -0- ---- - SC1 is
                GPIO
264             ; ---- -0- ---- - SC0 is
                GPIO
265
266 ;see errata for mask 4H80G, we find that GPIO pins need to be programmed to
                output
267 ; anyway. this is probably why our shit's not working OTL.
268
269 ESSI1_PRRD      EQU      $00003f      ; 0000 0000 0000 0000 0011 1111
270                                     ; 0000 0000 0000 0000 00-- ---- reserved
271                                     ; ---- -11 1111 all pins
                output
272
273
274 ; to write a command to the WM8741 is as simple as moving in the appropriate
                command
275 ; to ESSI1 TX0 and sleeping appropriately to ensure transmission.
276
277 ;these shifts are based on the fact that ESSI takes left-aligned data

```



```

278 DAC_ADDR_SHIFT EQU 9+8 ;we shift 9 bits over as our addresses comprise
279 ; the first 7 bits of our data word
280 DAC_DATA_SHIFT EQU 8 ;the data goes in the last 8 bits of the 16 bit
word
281
282 ;here are a bunch of appropriately shifted and masked addresses to use with our
system
283 DAC_LLSB_ADDR EQU ($0<<DAC_ADDR_SHIFT)
284 DAC_LMSB_ADDR EQU ($1<<DAC_ADDR_SHIFT)
285 DAC_RLSB_ADDR EQU ($2<<DAC_ADDR_SHIFT)
286 DAC_RMSB_ADDR EQU ($3<<DAC_ADDR_SHIFT)
287 DAC_VOL_ADDR EQU ($4<<DAC_ADDR_SHIFT)
288 DAC_FORMAT_ADDR EQU ($5<<DAC_ADDR_SHIFT)
289 DAC_FILTER_ADDR EQU ($6<<DAC_ADDR_SHIFT)
290 DAC_MODE1_ADDR EQU ($7<<DAC_ADDR_SHIFT)
291 DAC_MODE2_ADDR EQU ($8<<DAC_ADDR_SHIFT)
292 DAC_RESET_ADDR EQU ($9<<DAC_ADDR_SHIFT)
293 DAC_ADDCTL_ADDR EQU ($20<<DAC_ADDR_SHIFT)
294
295 ;here are a bunch of values we expect to initialize with
296 DAC_VOL_CONTROL EQU ($07<<DAC_DATA_SHIFT)
297 ; 0001 0111
298 ; 0--- ---- not muted
299 ; -00- ---- ZFLAG from both channels
300 ; ---0 ---- no analog mute on ZFLAG
301 ; ---- 0--- no mute
302 ; ---- -1-- control both channels w/
left atten
303 ; ---- --1- anti-clipping enable
304 ; ---- ---1 ramp vol changes
305 DAC_FILTER_CONTROL EQU ($01<<DAC_DATA_SHIFT)
306 ; 0000 0011
307 ; 0--- ---- normal ZFLAG
308 ; -00- ---- de-emph off
309 ; ---0 0--- DSD filter (don't care)
310 ; ---- -001 filter #2: minimum-phase
linear
311
312 DAC_MODE_CONTROL_1 EQU ($20<<DAC_DATA_SHIFT)
313 ; 0010 0000
314 ; 0--- ---- no 8fs mode
315 ; -01- ---- medium sample rate (96KHz) -
192 can be unstable
316 ; ---0 00-- autodetect MCLK
317 ; ---- --00 PCM mode
318
319 ;volume control stuff
320 ATTEN_MASK EQU $3ff ;WM8741 has space for 10 bits of volume
321 ATTEN_REG_MASK EQU $1f ;5 bits in the LSB of two registers
322 ATTEN_LATCH_MASK EQU $20 ;this bit latches all volume changes
323 ATTEN_STEP EQU 16 ;step in 2dB increments
324 MIN_ATTEN EQU 0 ;no attenuation == minimum
325 MAX_ATTEN EQU $3ff ;max attenuation = -127.5/mute
326 INIT_ATTEN EQU $dc ; ~-25 db to start, reasonable volume
327

```

```

1      opt      noidw
2      opt so,xr
3      page 132,66,3,3
4
5      ;
6      ; sound.asm
7      ;
8      ; This file offers interfacing code to the sound subsystem: the MP3 decoder and
9      ; the DAC.
10     ;
11
12     section sound
13     INCLUDE 'DSP_EQU.inc'
14     INCLUDE 'sound.inc'
15     INCLUDE 'macros.inc'
16     INCLUDE 'general.inc'
17     org      x:
18
19     ;VOLATILE declarations go here
20
21     ; sound_atten is our volume attenuation in 10 bits, in -0.125dB steps.
22     sound_atten:
23         ds      1
24
25
26     ;here are all our pointers/etc for the audio update system
27     curbuf:
28         ds      1                ;a pointer to the current buffer we're playing
29     curbuflen:
30         ds      1                ;length of the current buffer
31
32     nextbuf:
33         ds      1                ;a pointer to the next buffer (double
34         buffering)
35     nextbuflen:
36         ds      1                ;length of the next buffer
37
38     usedbuf:
39         ds      1                ;a pointer to a used buffer
40
41     mp3_kickstart:
42         ds      1                ;this is set non-zero if the mp3 decoder can
43                                   ; accept data, and was idle previously
44
45     ;CONSTANT declarations go here
46
47     org      p:                ;we want to be located in program space
48
49     ;PROGRAM (function) declarations go here
50
51     ; Fsetup_mp3
52     ; Functional Specification
53     ;
54     ; Description:      This function inits the MP3 decoder chip, synchronously.
55     ; It should be run when no interrupts are enabled.
56     ;

```

```

56 ; Operation:      This function sets up ESSIO to interface with the MP3
57 ; decoder chip (VLSI solutions VS1053b) in New Mode, and sets the clock
58 ; registers as appropriate. It also enables I2S output so we can get
59 ; our DAC outputting something.
60 ;
61 ; Arguments:      None.
62 ;
63 ; Return Values:  None.
64 ;
65 ; Global Variables: None.
66 ;
67 ; Shared Variables: None.
68 ;
69 ; Local Variables: None.
70 ;
71 ; Inputs:         None.
72 ;
73 ; Outputs:        None.
74 ;
75 ; Error Handling:  None.
76 ;
77 ; Algorithm:      None.
78 ;
79 ; Data Structures: None.
80 ;
81 ; Registers changed: None.
82 ;
83 ; Limitations:    None known.
84 ;
85 ; Known Bugs:     None.
86 ;
87 ; Special Notes:
88 ;
89 ; Revision History: 2011/02/16      Raymond Jimenez      first revision
90 ;
91     global  Fsetup_mp3
92 Fsetup_mp3:                ;C entry point
93
94     ;standard C prologue
95     move    #0,n6          ;k is the amount of local space
96     move    ssh,x:(r6)+    ;save the return address
97     move    (r6)+n6        ;allocate local stack space of size k
98
99
100    ;save all registers we use so we can restore later
101    move     r0,ssh
102    move     r1,ssl
103    PUSH_ACC a
104
105
106    ;initialize the ESSIO control registers
107    move     #>ESSIO_CRA_1ST,r0
108    move     r0,x:M_CRA0
109    move     #>ESSIO_CRB_1ST,r0
110    move     r0,x:M_CRB0
111
112    ;now configure the ESSIO pins as active and not GPIO

```

```

113     move    #>ESSI0_PCRC_SCI,r0
114     move    r0,x:M_PCRC
115     move    #>ESSI0_PRRC,r0
116     move    r0,x:M_PRRC
117     move    #>ESSI0_PDRC,r0
118     move    r0,x:M_PDRC
119
120     ;we load first data
121     move    #>$ffff00,r0
122     move    r0,x:M_TX00
123     ; (TX02 control XCS)
124     move    #XCS_HIGH,r0
125     move    r0,x:M_TX02
126
127     ;now enable transmitters
128     bset    #M_SSTE0,x:M_CRB0
129     bset    #M_SSTE2,x:M_CRB0
130     bset    #M_SSRE,x:M_CRB0
131
132     ;we wait for mp3 to become ready again
133     move    #>SET_CLOCK_WAIT,r0
134     jsr     sleep
135
136     ;we now need to do several things:
137     ; 1) set the clock multiplier on the MP3 to to 3 or so
138     ;     (12MHz clock is really slow)
139     ; 2) set the I2S output to enable
140     ; 3) set volume to max (DAC handles volume)
141     ; 4) in the future we'll add patches (like FLAC decoder)
142
143     ; 1) we set the clock multiplier
144     move    #>(SCI_WRITE_CMD|SCI_CLOCKF),r0
145     move    #>SCI_CLOCKF_VALUE,r1
146     jsr     write_sync_mp3_sci
147
148     ;we wait for mp3 to become ready again
149     move    #>SET_CLOCK_WAIT,r0
150     jsr     sleep
151
152     ;we can now use faster clock speeds, so set it
153
154
155     nop
156     nop
157     nop
158     jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted, we disable then
159                                     ;re-enable
160
161     move    #>ESSI0_CRA_SCI,r0    ;set new values of CRB/CRA
162     move    r0,x:M_CRA0
163     move    #>ESSI0_CRB_SCI,r0
164     move    r0,x:M_CRB0
165
166     ;as before, we now load shill data:
167     move    #>$ffff00,r0
168     move    r0,x:M_TX00
169     move    #XCS_HIGH,r0

```

```

170     move     r0,x:M_TX02
171
172     ;now enable transmitters
173     bset     #M_SSTE0,x:M_CRB0
174     bset     #M_SSTE2,x:M_CRB0
175     bset     #M_SSRE,x:M_CRB0
176
177     ; 2) enable I2S output
178
179     ;to enable I2S, we turn GPIO to output and turn on I2S
180     ;set GPIO flags:
181     move     #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0           ;we need to set where
182                                                         ; to write to in 1053
183     move     #>GPIO_DDR_ADDR,r1
184     jsr      write_sync_mp3_sci
185     move     #>SET_WRAMADDR_WAIT,r0
186     jsr      sleep
187
188     move     #>(SCI_WRITE_CMD|SCI_WRAM),r0               ;now write content
189     move     #>GPIO_DDR_VALUE,r1
190     jsr      write_sync_mp3_sci
191     move     #>SET_WRAM_WAIT,r0
192     jsr      sleep
193
194     ;set I2S enable:
195     move     #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0           ;we need to set where
196                                                         ; to write to in 1053
197     move     #>I2S_CONFIG_ADDR,r1
198     jsr      write_sync_mp3_sci
199     move     #>SET_WRAMADDR_WAIT,r0
200     jsr      sleep
201
202     move     #>(SCI_WRITE_CMD|SCI_WRAM),r0               ;now write content
203     move     #>I2S_CONFIG_VALUE,r1
204     jsr      write_sync_mp3_sci
205     move     #>SET_WRAM_WAIT,r0
206     jsr      sleep
207
208
209     ; 3) set analog out to max volume
210     move     #>(SCI_WRITE_CMD|SCI_VOL),r0
211     move     #>SCI_VOL_VALUE,r1
212     jsr      write_sync_mp3_sci
213     move     #>SET_VOL_WAIT,r0
214     jsr      sleep
215
216
217     ; 4) instead of patches, we set SDI test mode for sine waves.
218     ;we first set SM_TESTS = 1
219     move     #>(SCI_WRITE_CMD|SCI_MODE),r0
220     move     #SCI_MODE_VALUE,r1
221     jsr      write_sync_mp3_sci
222     move     #>SET_MODE_WAIT,r0
223     jsr      sleep
224
225     ;then write AIADDR to enable the sine wave test
226     ;     move     #>(SCI_WRITE_CMD|SCI_AIADDR),r0

```

```

227 ;      move    #SCI_AIADDR_VALUE,r1
228 ;      jsr     write_sync_mp3_sci
229 ;      move    #>SET_AIADDR_WAIT,r0
230 ;      jsr     sleep
231
232 ; 5) we now switch to data mode
233 jsr     mp3_to_data_mode
234
235 ;and we setup the DMA control registers as we'll be using
236 ; them in the future
237 move    #>DCR2,r0
238 move    r0,x:M_DCR2
239 move    #>M_TX00,r0
240 move    r0,x:M_DDR2
241
242 ;we also install the interrupt handlers we'll be using
243
244 move    #>$0bf080,r0      ;unconditional branch
245 move    r0,p:I_DMA2
246 move    #end_mp3_dma_handler,r1 ;install DMA handler
247 move    r1,p:(I_DMA2+1)
248
249 ;install the pair of DREQ handlers
250 move    #dreq_asserted_handler,r1
251 move    r0,p:I_IRQB
252 move    r1,p:(I_IRQB+1)
253
254 move    #dreq_deasserted_handler,r1
255 move    r0,p:I_IRQC
256 move    r1,p:(I_IRQC+1)
257
258 ;we now clear XCS
259 move    #0,r1
260
261 ;this commented out portion implements a SDI test vector (5133Hz sine)
262
263 ;      nop      ;wait for delay
264 ;      nop
265 ;      nop
266
267 ;      move     #>$53ef,r1
268 ;      move     r1,x:M_TX00      ;shill write to TX00
269 ;      nop      ;wait for delay
270 ;      nop
271 ;      nop
272 ;      jclr     #M_TDE,x:M_SSISR0,* ;wait until transmitted
273
274 ;      move     #>$6e7e,r1
275 ;      move     r1,x:M_TX00      ;shill write to TX00
276 ;      nop      ;wait for delay
277 ;      nop
278 ;      nop
279 ;      jclr     #M_TDE,x:M_SSISR0,* ;wait until transmitted
280
281 ;      move     #>$0,r1
282 ;      move     r1,x:M_TX00      ;shill write to TX00
283 ;      nop      ;wait for delay

```

```

284 ;      nop
285 ;      nop
286 ;      jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
287
288 ;      move    #>$0,r1
289 ;      move    r1,x:M_TX00            ;shill write to TX00
290 ;      nop     ;wait for delay
291 ;      nop
292 ;      nop
293 ;      jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
294
295
296
297
298
299 ;end of body code, now restore all registers
300     POP_ACC a
301     move    ssl,r1
302     move    ssh,r0
303
304     ;standard C epilogue
305     move    #0+1,n6
306     move    (r6)-n6            ;deallocate local stack space, set ccr flags
307     tst     a
308     move    x:(r6),ssh        ;get return address
309     nop
310     rts     ;return from subroutine
311
312 ;end Fsetup_mp3
313
314
315 ; mp3_to_data_mode
316 ; Functional Specification
317 ;
318 ; Description:      mp3_to_data_mode changes ESSIO to enter data
319 ; mode, so that we can transmit MP3 data to it.
320 ;
321 ; Operation:      mp3_to_data_mode changes the mode to enable 16-bit
322 ; data words and reconfigures PC2 to function as a frame sync, thus giving
323 ; us an easy time. It also disables the SC1 (XCS) transmitter and sets it
324 ; as a GPIO pin, making it easier for us.
325 ;
326 ; Arguments:      None.
327 ;
328 ; Return Values:  None.
329 ;
330 ; Global Variables: None.
331 ;
332 ; Shared Variables: None.
333 ;
334 ; Local Variables: None.
335 ;
336 ; Inputs:        None.
337 ;
338 ; Outputs:       None.
339 ;
340 ; Error Handling: None.

```

```

341 ;
342 ; Algorithm:      None.
343 ;
344 ; Data Structures: None.
345 ;
346 ; Registers changed: None.
347 ;
348 ; Limitations:    None known.
349 ;
350 ; Known Bugs:     None.
351 ;
352 ; Special Notes:   This function turns off ESSIO's interrupts. It does not
353 ; enable the DREQ/IRQ1 interrupt.
354 ;
355 ; Revision History: 2011/03/28   Raymond Jimenez   first revision
356 ;
357     global mp3_to_data_mode
358 mp3_to_data_mode:                ;assembly entry point
359
360     ;save all registers we use so we can restore later
361     move    r0,ssh
362
363     ;we make sure all transmits are done:
364     nop
365     nop
366     nop
367     jclr    #M_TDE,x:M_SISR0,*    ;wait until transmitted
368
369     ;need to temporarily disable ESSIO pins otherwise they retain
370     ;functionality
371     move    #>0,r0
372     move    r0,x:M_PCRC
373
374     ;we then reconfigure ESSIO to give us automatic frame syncs
375     ; (auto XDCS)
376     move    #ESSIO_CRA_DATA,r0
377     move    r0,x:M_CRA0
378     move    #ESSIO_CRB_DATA,r0
379     move    r0,x:M_CRB0
380
381     ;then, we set the GPIO pins as necessary
382     move    #>ESSIO_PCRC_DATA,r0
383     move    r0,x:M_PCRC
384
385     ;load shll data
386     move    #>0,r0
387     move    r0,x:M_TX00
388
389     ;then enable transmitters
390     bset    #M_SSTE0,x:M_CRB0
391
392     ;and that's it
393
394     ;end of body code, now restore all registers
395     move    ssh,r0
396
397     nop    ;pipeline cache issue (see errata)

```



```

397         rts        ;return from subroutine
398 ;end mp3_to_data_mode
399
400 ; mp3_to_control_mode
401 ; Functional Specification
402 ;
403 ; Description:      mp3_to_control_mode changes ESSIO to control mode,
404 ; so we can use write_sync_mp3_sci/read_sync_mp3_sci.
405 ;
406 ; Operation:       mp3_to_control_mode changes the ESSIO/GPIO registers
407 ; such that we can use it to transmit control words now.
408 ;
409 ; Arguments:       None.
410 ;
411 ; Return Values:   None.
412 ;
413 ; Global Variables: None.
414 ;
415 ; Shared Variables: None.
416 ;
417 ; Local Variables: None.
418 ;
419 ; Inputs:          None.
420 ;
421 ; Outputs:         None.
422 ;
423 ; Error Handling:   None.
424 ;
425 ; Algorithm:        None.
426 ;
427 ; Data Structures: None.
428 ;
429 ; Registers changed: None.
430 ;
431 ; Limitations:      None known.
432 ;
433 ; Known Bugs:       None.
434 ;
435 ; Special Notes:    This function turns off ESSIO's interrupts. It does not
436 ; enable/disable the DREQ/IRQ1 interrupt.
437 ;
438 ; Revision History: 2011/03/28    Raymond Jimenez    first revision
439 ;
440     global mp3_to_control_mode
441 mp3_to_control_mode:                ;assembly entry point
442
443     ;save all registers we use so we can restore later
444     move    r0,ssh
445
446     ;we manually disable DMA.
447     bclr    #M_DE,x:M_DCR2
448     ;and we wait for the DMA transfers to finish
449     jset    #M_DACT,x:M_DSTR,*
450
451     ;we make sure all transmits are done:
452     nop
453     nop

```

```

454      nop
455      jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
456
457      ;need to temporarily disable ESSI pins otherwise they retain
      ;functionality
458      move    #>0,r0
459      move    r0,x:M_PCRC
460
461
462      ;we then reconfigure ESSI0 to give us automatic frame syncs
463      ; (auto XDCS)
464      move    #ESSI0_CRA_SCI,r0
465      move    r0,x:M_CRA0
466      move    #ESSI0_CRB_SCI,r0
467      move    r0,x:M_CRB0
468
469      ;first, we set the GPIO pins as necessary
470      move    #>ESSI0_PCRC_SCI,r0
471      move    r0,x:M_PCRC
472
473      ;we load first data
474      move    #>$ffff00,r0
475      move    r0,x:M_TX00
476      ; (TX02 control XCS)
477      move    #XCS_HIGH,r0
478      move    r0,x:M_TX02
479
480      bset    #M_SSTE0,x:M_CRB0
481      bset    #M_SSTE2,x:M_CRB0
482      bset    #M_SSRE,x:M_CRB0
483
484
485      ;and that's it
486
487      ;end of body code, now restore all registers
488      move    ssh,r0
489
490      nop      ;pipeline cache issue (see errata)
491      rts      ;return from subroutine
492 ;end mp3_to_control_mode
493
494
495 ; write_sync_mp3_sci
496 ; Functional Specification
497 ;
498 ; Description:      write_sync_mp3_sci transmits an SCI command to the mp3
499 ; decoder, synchronously. command is guaranteed to be transmitted but the
500 ; mp3 decoder is not guaranteed to be ready after this. (you must insert
501 ; a sleep after to account for worst-case wait, or wait for DREQ to become
    high)
502 ;
503 ; ESSI0 interrupts must be disabled before invocation.
504 ;
505 ; Operation:      write_sync_mp3_sci first changes the ESSI0 TX2 to indicate
506 ; XCS is selected. it then checks to make sure the TX buffers are empty, and
507 ; transmits the desired control words as necessary.
508 ;

```

```

509 ; Arguments:      r0 - SCI command and address to write, left aligned.
510 ;                  r0[23:16] - should be $02 for write
511 ;                  r0[15:8]  - SCI address
512 ;                  r0[7:0]   - reserved, write 0 for compat
513 ;
514 ;                  r1 - SCI data to write
515 ;                  r1[23:8]  - SCI data to write
516 ;                  r1[7:0]   - reserved, write 0 for compat
517 ;
518 ; Return Values:   None.
519 ;
520 ; Global Variables: None.
521 ;
522 ; Shared Variables: None.
523 ;
524 ; Local Variables: None.
525 ;
526 ; Inputs:          None.
527 ;
528 ; Outputs:          None.
529 ;
530 ; Error Handling:   None.
531 ;
532 ; Algorithm:        None.
533 ;
534 ; Data Structures:  None.
535 ;
536 ; Registers changed: None.
537 ;
538 ; Limitations:      None known.
539 ;
540 ; Known Bugs:       None.
541 ;
542 ; Special Notes:    We require the strange bitfield structure of r0/1 in order
543 ;                   to
544 ;                   prevent us from having to transfer into a, and then rotate. it's much faster
545 ;                   this way, and since we mostly transmit constants to SCI, we can just
546 ;                   pre-compose commands via macros.
547 ;
548 ; Revision History:  2011/02/16    Raymond Jimenez    first revision
549 ;
550 ;                   global write_sync_mp3_sci
551 write_sync_mp3_sci:                                     ;assembly entry point
552 ;
553 ;save all registers we use so we can restore later
554 PUSH_ACC a
555 move     r2,ssh
556 ;
557 ;note that these sections are time-critical, so we block on them.
558 ; (this is another reason to up the clock speed on our serial
559 ; transmits)
560 START_CRITICAL
561 ;we first make sure the transmitter is empty
562 jclr     #M_TDE,x:M_SSISR0,*      ;keep on jumping here until TDE is set
563 move     r0,x:M_TX00              ;transmit first 16 bits: command, address

```

```

564      ;we first want to make sure XCS is turned on for these words,
565      ; so set up TX10/TX20 appropriately
566      move    #XCS_HIGH,r2
567      move    r2,x:M_TX02
568
569
570      ;we first make sure the transmitter is empty
571
572      ;we have a two-cycle pipeline delay after we write these registers
573      ; see DSP3609 user manual, pg 7-31
574      nop     ;wait for delay
575      nop
576      nop
577      jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
578      move    r0,x:M_TX00            ;transmit first 16 bits: command, address
579
580      ;we first want to make sure XCS is turned on for these words,
581      ; so set up TX10/TX20 appropriately
582      move    #XCS_LOW,r2
583      move    r2,x:M_TX02
584
585
586      ;and then we move in our words:
587
588      nop     ;wait for delay
589      nop
590      nop
591      jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
592
593      move    r1,x:M_TX00            ;next 16 bits, also filling TX10/TX20
594      move    #XCS_LOW,r2
595      move    r2,x:M_TX02
596
597
598
599      ;we now clear XCS
600
601      nop     ;wait for delay
602      nop
603      nop
604      jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
605
606      move    r1,x:M_TX00            ;shll write to TX00
607
608      move    #XCS_HIGH,r2
609      move    r2,x:M_TX02
610
611      END_CRITICAL
612
613      ;end of body code, now restore all registers
614      move    ssh,r2
615      POP_ACC a
616
617      nop     ;pipeline cache issue (see errata)
618      rts     ;return from subroutine
619 ;end write_sync_mp3_sci
620

```

```

621
622 ; read_sync_mp3_sci
623 ; Functional Specification
624 ;
625 ; Description:      read_sync_mp3_sci reads from the mp3 decoder, selecting a
626 ; register/address to read from.
627 ;
628 ; ESSIO interrupts must be disabled before invocation.
629 ;
630 ; Operation:        write_sync_mp3_sci first
631 ;
632 ; Arguments:         r0 - SCI command and address to write, left aligned.
633 ;                   r0[23:16] - should be $03 for read
634 ;                   r0[15:8]  - SCI address
635 ;                   r0[7:0]   - reserved, write 0 for compat
636 ;
637 ; Return Values:     a1 - data word transmitted by mp3 decoder
638 ;
639 ; Global Variables:  None.
640 ;
641 ; Shared Variables:  None.
642 ;
643 ; Local Variables:   None.
644 ;
645 ; Inputs:            None.
646 ;
647 ; Outputs:           None.
648 ;
649 ; Error Handling:    None.
650 ;
651 ; Algorithm:         None.
652 ;
653 ; Data Structures:   None.
654 ;
655 ; Registers changed:  a
656 ;
657 ; Limitations:       None known.
658 ;
659 ; Known Bugs:        None.
660 ;
661 ; Special Notes:     We require the strange bitfield structure of r0/1 in order
662 ; to
663 ; prevent us from having to transfer into a, and then rotate. it's much faster
664 ; this way, and since we mostly transmit constants to SCI, we can just
665 ; pre-compose commands via macros.
666 ;
667 ; Revision History:  2011/03/27   Raymond Jimenez   first revision
668 ;
669     global read_sync_mp3_sci
670 read_sync_mp3_sci:
671     ;save all registers we use so we can restore later
672     move     r2,ssh
673
674
675     ;note that these sections are time-critical, so we block on them.
676     ; (this is another reason to up the clock speed on our serial

```

```

676     transmits)
677     START_CRITICAL
678
679     ;we first make sure the transmitter is empty
680     jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
681     move    r0,x:M_TX00            ;transmit first 16 bits: command, address
682
683     ;we first want to make sure XCS is turned on for these words,
684     ; so set up TX10/TX20 appropriately
685     move    #XCS_HIGH,r2
686     move    r2,x:M_TX02
687
688
689     ;we first make sure the transmitter is empty
690
691     ;we have a two-cycle pipeline delay after we write these registers
692     ; see DSP3609 user manual, pg 7-31
693     nop     ;wait for delay
694     nop
695     nop
696     jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
697     move    r0,x:M_TX00            ;transmit first 16 bits: command, address
698
699     ;we first want to make sure XCS is turned on for these words,
700     ; so set up TX10/TX20 appropriately
701     move    #XCS_LOW,r2
702     move    r2,x:M_TX02
703
704
705     ;and then we move in our words:
706
707     nop     ;wait for delay
708     nop
709     nop
710     jclr    #M_TDE,x:M_SSISR0,*    ;wait until transmitted
711
712     move    r0,x:M_TX00            ;shll write to get data; don't care
713                                     ; what data we transmit.
714     move    #XCS_LOW,r2
715     move    r2,x:M_TX02
716
717
718     nop     ;wait for delay
719     nop
720     nop
721     jclr    #M_TDE,x:M_SSISR0,*    ;keep on jumping here until TDE is set
722
723
724     move    x:M_RX0,a              ;read recieved data - we expect it to be
725                                     ; read as it's completed so it doesn't
726                                     ; get overwritten. (right after TDE
727                                     ; is guaranteed to be one serial word)
728
729
730
731     ;we now clear XCS

```

```

732         jclr      #M_TDE,x:M_SISR0,*    ;wait until transmitted
733
734         move       r1,x:M_TX00           ;shill write to TX00
735
736         move       #XCS_HIGH,r2
737         move       r2,x:M_TX02
738
739         END_CRITICAL
740
741         ;end of body code, now restore all registers
742         move       ssh,r2
743
744         nop        ;pipeline cache issue (see errata)
745         rts        ;return from subroutine
746 ;end read_sync_mp3_sci
747
748
749 ; Faudio_play
750 ; Functional Specification
751 ;
752 ; Description:      Faudio_play sets up the interrupts and appropriate
variables
753 ; for the audio playback system.
754 ;
755 ; Operation:        Faudio_play first calls Faudio_halt, which synchronously
756 ; stops audio playback. We then set up the registers again, and then start
757 ; the file playing by setting up the interrupts and letting the ESSIO TX empty
758 ; trigger run the DMA request.
759 ;
760 ; Arguments:        a1 - pointer to new buffer to use
761 ;                  b1 - length of buffer at pointer
762 ;
763 ; Return Values:    None.
764 ;
765 ; Global Variables: None.
766 ;
767 ; Shared Variables: None.
768 ;
769 ; Local Variables:  None.
770 ;
771 ; Inputs:           None.
772 ;
773 ; Outputs:          None.
774 ;
775 ; Error Handling:   None.
776 ;
777 ; Algorithm:        None.
778 ;
779 ; Data Structures:  None.
780 ;
781 ; Registers changed: None.
782 ;
783 ; Limitations:      None known.
784 ;
785 ; Known Bugs:       None.
786 ;
787 ; Special Notes:

```

```

788 ;
789 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
790 ;
791     global  Faudio_play
792 Faudio_play:                ;C entry point
793
794     ;standard C prologue
795     move    #0,n6            ;k is the amount of local space
796     move    ssh,x:(r6)+      ;save the return address
797     move    (r6)+n6          ;allocate local stack space of size k
798
799
800     ;save all registers we use so we can restore later
801     move    r0,ssh
802     move    r1,ssl
803     PUSH_ACC b
804
805     ;we first halt any playback that might be happening
806 ;     jsr    Faudio_halt
807
808     ;we are now guaranteed that the VLSI can take our data, DREQ must be
    high
809
810     ;now, we need to setup the DMA control registers and our buffers
811     move    #>NO_BUFFER_LEN,r0
812     move    r0,x:nextbuf      ;we clear out these variables to prevent
    confusion
813     move    r0,x:nextbuflen
814     move    r0,x:usedbuf
815
816     move    a1,x:curbuf        ;and move in our actual buffer values
817     move    b1,x:curbuflen
818
819     ;process the mp3 buffer
820     move    a1,r0 ;prepare arguments for prepare_mp3_buffer
821     move    b1,r1
822     jsr    process_mp3_buffer
823
824     move    #>NO_KICKSTART,r0
825     move    r0,x:mp3_kickstart ;we're starting the writing process, so no
    ;kickstart
826
827
828     move    x:curbuf,a         ;and move in our actual buffer values
829     move    x:curbuflen,b
830
831     move    a1,x:M_DSR2        ;we now setup the DMA pointers
832     ;unfortunately, DMA transfers counter + 1 in total, so we subtract:
833     sub     #>1,b
834     move    b1,x:M_DC02        ;tell the DMA how many words from this buffer
835
836
837     ;before DMA can go, we need to set enable the interrupts on DREQ off/on
838     move    x:M_IPRC,b         ;we need to OR in our new priority bits
839     and     #>((~M_IBL)&(~M_ICL)&(~M_D2L)),b ;clear out bits there
840     or      #>(DREQ_CTL_PRIO<<M_IBL0)|(DREQ_CTL_PRIO<<M_ICL0)|(3<<M_D2L0),b
841
842                                     ;set our new bits

```



```

843         move    b1,x:M_IPRC
844
845         ;and then finally, we enable the DMA and get going
846
847         bset     #M_DE,x:M_DCR2 ;set enable, since DMA is DE-enabled
848
849         ;at this point we should be going at full tilt:
850         ; if the buffer runs out, the DMA done interrupt will handle it
851         ; if DREQ goes low, the DREQ-done handler will disable DMA
852         ; if DREQ goes high after low, the DREQ-more handler will enable DMA
853
854         ;end of body code, now restore all registers
855         POP_ACC b
856         move     ssl,r1
857         move     ssh,r0
858
859         ;standard C epilogue
860         move     #0+1,n6
861         move     (r6)-n6          ;deallocate local stack space, set ccr flags
862         tst      a
863         move     x:(r6),ssh      ;get return address
864
865         nop      ;pipeline cache issue (see errata)
866         rts      ;return from subroutine
867
868 ;end Faudio_play
869
870 ; Faudio_halt
871 ; Functional Specification
872 ;
873 ; Description:      Faudio_halt turns off the DMA and other interrupts
874 ; so that we can tell the decoder to halt, synchronously. This function does
875 ; not return until audio playback has stopped.
876 ;
877 ; Operation:       Faudio_halt follows the procedure laid out in the 1053
878 ; manual: it first reads endFillByte, sends 2052 bytes of endFillByte,
879 ; then sets SM_CANCEL, sends another 2052 bytes of endFillByte, and then
880 ; returns.
881 ;
882 ; Arguments:       None.
883 ;
884 ; Return Values:   None.
885 ;
886 ; Global Variables: None.
887 ;
888 ; Shared Variables: None.
889 ;
890 ; Local Variables: None.
891 ;
892 ; Inputs:          None.
893 ;
894 ; Outputs:         None.
895 ;
896 ; Error Handling:  None.
897 ;
898 ; Algorithm:       None.
899 ;

```

```

900 ; Data Structures:  None.
901 ;
902 ; Registers changed: None.
903 ;
904 ; Limitations:      None known.
905 ;
906 ; Known Bugs:      None.
907 ;
908 ; Special Notes:
909 ;
910 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
911 ;
912     global  Faudio_halt
913 Faudio_halt:                ;C entry point
914
915     ;standard C prologue
916     move    #0,n6            ;k is the amount of local space
917     move    ssh,x:(r6)+      ;save the return address
918     move    (r6)+n6          ;allocate local stack space of size k
919
920
921     ;save all registers we use so we can restore later
922     PUSH_ACC a
923     PUSH_ACC b
924     move    r0,ssh
925     move    r1,ssl
926     move    x0,ssh
927
928
929     ;disable interrupts, all reads from here on are synchronous
930     move    x:M_IPRC,a        ;move to a so we can clear out bits
931     and     #>((~M_IBL)&(~M_ICL)&(~M_D2L)),a ;clear out bits there
932                                     (IRQB/IRQC are DRQ)
933     move    a1,x:M_IPRC
934
935     ;we switch to control mode to read endFillByte
936     jsr     mp3_to_control_mode
937
938     move    #>(SCI_WRITE_CMD|SCI_WRAMADDR),r0        ;set address
939     move    #>ENDFILLBYTE_ADDR,r1
940     jsr     write_sync_mp3_sci
941     move    #>SET_WRAMADDR_WAIT,r0
942     jsr     sleep
943
944     move    #>(SCI_READ_CMD|SCI_WRAM),r0              ;read data
945     jsr     read_sync_mp3_sci
946     ;we now have endFillByte in a1[15-8]. let's duplicate it
947     ; to [7-0] for ease of use.
948     and     #>$00ff00,a                                ;make sure only one byte is set
949     move    a1,b                                ;we use b as a scratch register
950     lsl     #8,b                                ;put a copy in the top byte
951     move    b1,x0
952     or      x0,a                                ;now a has a copy of two bytes of
953     endfillbyte
954
955     jsr     mp3_to_data_mode                ;now we need to write 2052 bytes

```

```

955
956     do        #2052,write_endfillbyte
957     move      a1,x:M_TX00          ;write endfillbyte
958     nop       ;account for pipeline delay
959     nop
960     nop
961     jclr      #M_TDE,x:M_SSISR0,*   ;keep on jumping here until transmitted

962     nop       ;spacing nops
963     nop
964     nop
965 write_endfillbyte:
966
967     jsr       mp3_to_control_mode    ;we now need to shift to control mode to
968                                     ; set SM_CANCEL
969     move      #>(SCI_WRITE_CMD|SCI_MODE),r0
970     move      #>SCI_MODE_CANCEL_VALUE,r1
971     jsr       write_sync_mp3_sci
972     move      #>SET_MODE_WAIT,r0
973     jsr       sleep
974     ;now that we've written cancel, we transmit 2052 bytes of endfillbyte
    again
975
976     jsr       mp3_to_data_mode
977
978     do        #2052,write_endfillbyte2
979     move      a1,x:M_TX00          ;write endfillbyte
980     nop       ;account for pipeline delay
981     nop
982     nop
983     jclr      #M_TDE,x:M_SSISR0,*   ;keep on jumping here until transmitted

984     nop       ;spacing nops
985     nop
986     nop
987 write_endfillbyte2:
988
989     ;we're now done.
990
991 ;end of body code, now restore all registers
992     move      ssh,x0
993     move      ssl,r1
994     move      ssh,r0
995     POP_ACC   b
996     POP_ACC   a
997
998
999     ;standard C epilogue
1000     move      #0+1,n6
1001     move      (r6)-n6              ;deallocate local stack space, set ccr flags
1002     tst       a
1003     move      x:(r6),ssh          ;get return address
1004
1005     nop       ;pipeline cache issue (see errata)
1006     rts       ;return from subroutine
1007
1008 ;end Faudio_halt

```

```

1009
1010 ; end_mp3_dma_handler
1011 ; Functional Specification
1012 ;
1013 ; Description:      end_mp3_dma_handler takes care of switching to the next
1014 ; buffer at the end of a DMA transfer to the mp3 decoder.
1015 ;
1016 ; Operation:        end_mp3_dma_handler handles the case when we get to the end
1017 ; of a DMA transfer, and so we need more data. We first check to see if we have
1018 ; a valid next buffer; if so, we swap it out and go for it.
1019 ;
1020 ; if there is no valid next data buffer, we set mp3_kickstart to
NEEDS_KICKSTART
1021 ; and we indicate that our buffer has been used up by moving it into usedbuf.
1022 ;
1023 ; Arguments:         None.
1024 ;
1025 ; Return Values:     None.
1026 ;
1027 ; Global Variables:  None.
1028 ;
1029 ; Shared Variables:  None.
1030 ;
1031 ; Local Variables:   None.
1032 ;
1033 ; Inputs:             None.
1034 ;
1035 ; Outputs:            None.
1036 ;
1037 ; Error Handling:     None.
1038 ;
1039 ; Algorithm:          None.
1040 ;
1041 ; Data Structures:   None.
1042 ;
1043 ; Registers changed:  None.
1044 ;
1045 ; Limitations:        None known.
1046 ;
1047 ; Known Bugs:         None.
1048 ;
1049 ; Special Notes:
1050 ;
1051 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
1052 ;
1053     global end_mp3_dma_handler
1054 end_mp3_dma_handler:                ;assembly entry point
1055
1056     ;save all registers we use so we can restore later
1057     PUSH_ACC a
1058     move x0,ssh
1059     move r0,ssl
1060
1061     ;we first clear out interrupt enable bit to prevent
1062     ; multiple interrupts
1063     bclr    #M_DIE,x:M_DCR2          ;clear out interrupt bit

```

```

1064
1065     ;since we're called if DE has also been hand-deasserted
1066     ; (as in DREQ stopping our data transfer), we
1067     ; check to see if we were at the end of the transfer by comparing
1068     ; the buffer length to DCR+1.
1069     move    x:M_DC02,a
1070     add     #>1,a                ;number of transfer done is DCR+1, so we
1071                                     ; compensated for this
1072     move    x:curbuflen,x0
1073     cmp     x0,a
1074
1075     jeq     end_mp3_true_end      ;if we got here by an artificial DE
1076                                     ;disable, we skip our next-buffer logic
1077 end_mp3_false_end:
1078     ;we just set the DIE bit and go on our way, if false
1079
1080     ;re-enable DMA channel interrupts, in all cases
1081     bset    #M_DIE,x:M_DCR2
1082     jmp     end_mp3_dma_handler_done
1083
1084 end_mp3_true_end:
1085     ;we're now sure that we're operating from the end of the buffer.
1086
1087     move    x:curbuf,r0           ;we indicate this buffer's been used
1088     nop
1089     move    r0,x:usedbuf
1090
1091     ;let's check if there's more data in the next buffer.
1092     move    x:nextbuflen,a
1093     cmp     #>NO_BUFFER_LEN,a    ;check if we have no buffer
1094     jeq     end_mp3_dma_no_buf
1095
1096 end_mp3_dma_buf: ;we're now guaranteed to have a buffer to move in
1097     move    x:nextbuf,r0
1098     nop
1099     move    r0,x:M_DSR2           ;load the new source buffer
1100     move    r0,x:curbuf
1101
1102     move    x:nextbuflen,a        ;we now subtract one to get DC0
1103     move    a1,x:curbuflen
1104     sub     #>1,a
1105     move    a1,x:M_DC02           ;load in the new counter
1106
1107     move    #>NO_BUFFER_LEN,r0    ;and we mark that we have no next
1108     move    r0,x:nextbuf
1109     move    r0,x:nextbuflen
1110
1111     ;also, we mark that we don't need a kickstart, we've done it
1112     move    #>NO_KICKSTART,r0
1113     move    r0,x:mp3_kickstart
1114
1115     ;we now enable the DMA channel again
1116     bset    #M_DIE,x:M_DCR2
1117     bset    #M_DE,x:M_DCR2
1118
1119     jmp     end_mp3_dma_handler_done
1120

```

```

1121 end_mp3_dma_no_buf: ;we're guaranteed to have no buffers to move in
1122     ;since we have no next buffer, we mark the current buffer
1123     ; used as well.
1124     move    x:curbuf,r0
1125     nop
1126     move    r0,x:usedbuf
1127
1128     ;we invalidate the current buffer by setting it to NO_BUFFER_LEN
1129     move    a1,x:curbuflen           ;contains NO_BUFFER_LEN
1130
1131     ;we will need a kickstart after this, so we set it
1132     move    #>NEEDS_KICKSTART,r0
1133     move    r0,x:mp3_kickstart
1134
1135     ;that should be it!
1136     ;re-enable DMA channel interrupts, in all cases
1137     bset    #M_DIE,x:M_DCR2
1138
1139
1140 end_mp3_dma_handler_done:
1141
1142     ;end of body code, now restore all registers
1143     move    ssl,r0
1144     move    ssh,x0
1145     POP_ACC a
1146
1147     nop     ;pipeline cache issue (see errata)
1148     rti     ;return from interrupt
1149 ;end end_mp3_dma_handler
1150
1151 ; dreq_asserted_handler
1152 ; Functional Specification
1153 ;
1154 ; Description:      dreq_asserted_handler takes care of the situation when
1155 ; the 1053's DREQ has been freshly asserted.
1156 ;
1157 ; Operation:        dreq_asserted_handler sets the DMA enable bit on our
1158 ; DMA channel. we only see a negative-going edge on DREQ if we are newly
1159 ; renewing an old data transfer, because DREQ should start high coming out
1160 ; of control mode.
1161 ;
1162 ; Arguments:         None.
1163 ;
1164 ; Return Values:     None.
1165 ;
1166 ; Global Variables:  None.
1167 ;
1168 ; Shared Variables:  None.
1169 ;
1170 ; Local Variables:   None.
1171 ;
1172 ; Inputs:            None.
1173 ;
1174 ; Outputs:           None.
1175 ;
1176 ; Error Handling:     None.
1177 ;

```

```

1178 ; Algorithm:      None.
1179 ;
1180 ; Data Structures: None.
1181 ;
1182 ; Registers changed: None.
1183 ;
1184 ; Limitations:     None known.
1185 ;
1186 ; Known Bugs:      None.
1187 ;
1188 ; Special Notes:
1189 ;
1190 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
1191 ;
1192     global dreq_asserted_handler
1193 dreq_asserted_handler:                ;assembly entry point
1194
1195     ;save all registers we use so we can restore later
1196     PUSH_ACC a
1197
1198     ;we check if there's good data to be transmitted
1199     move    x:curbuflen,a
1200     cmp     #>NO_BUFFER_LEN,a
1201     jeq     dreq_asserted_no_buffer
1202
1203 dreq_asserted_buffer:
1204     bset    #M_DE,x:M_DCR2    ;set our DMA channel to go
1205     jmp     dreq_asserted_done
1206
1207 dreq_asserted_no_buffer:
1208     move    #>NEEDS_KICKSTART,a
1209     move    a1,x:mp3_kickstart
1210
1211 dreq_asserted_done:
1212     ;end of body code, now restore all registers
1213     POP_ACC a
1214
1215     nop     ;pipeline cache issue (see errata)
1216     rti     ;return from interrupt
1217
1218 ;end dreq_asserted_handler
1219
1220 ; dreq_deasserted_handler
1221 ; Functional Specification
1222 ;
1223 ; Description:      dreq_deasserted_handler takes care of the case when the
1224 1053 ; can't receive any more data.
1225 ;
1226 ; Operation:        It disables the DMA channel by clearing DE.
1227 ;
1228 ; Arguments:        None.
1229 ;
1230 ; Return Values:    None.
1231 ;
1232 ; Global Variables: None.
1233 ;

```

```

1234 ; Shared Variables: None.
1235 ;
1236 ; Local Variables: None.
1237 ;
1238 ; Inputs: None.
1239 ;
1240 ; Outputs: None.
1241 ;
1242 ; Error Handling: None.
1243 ;
1244 ; Algorithm: None.
1245 ;
1246 ; Data Structures: None.
1247 ;
1248 ; Registers changed: None.
1249 ;
1250 ; Limitations: None known.
1251 ;
1252 ; Known Bugs: None.
1253 ;
1254 ; Special Notes:
1255 ;
1256 ; Revision History: 2011/02/16 Raymond Jimenez first revision
1257 ;
1258     global dreq_deasserted_handler
1259 dreq_deasserted_handler:                ;assembly entry point
1260
1261     ;save all registers we use so we can restore later
1262
1263     bclr    #M_DE,x:M_DCR2                ;disable the current DMA
1264     nop
1265     nop
1266     nop ;pipeline delays
1267     ;wait for it to finish its current word transfer
1268 ;     jset    #M_DACT,x:M_DSTR,*
1269
1270     ;end of body code, now restore all registers
1271
1272     nop ;pipeline cache issue (see errata)
1273     rti ;return from interrupt
1274 ;end dreq_deasserted_handler
1275
1276
1277 ; Fupdate
1278 ; Functional Specification
1279 ;
1280 ; Description: Fupdate is to be called periodically, and gives new data
1281 ; to the DMA if necessary. it returns true (non-zero) if more data was needed
1282 ; and the passed buffer is a used buffer. it returns false (zero) if no more
1283 ; data can be taken.
1284 ;
1285 ; Operation:
1286 ;
1287 ; we check: is nextbuf occupied?
1288 ; yes:
1289 ; case #1: curbuf is occupied, nextbuf occupied, anybuf passed

```



```

1290 ; return false -- no space
1291 ;
1292 ; no: we then check, is curbuf occupied?
1293 ;
1294 ; no:
1295 ; case #2: curbuf is empty, nextbuf is empty, usedbuf (formery curbuf) is
    passed
1296 ; -- this is the kickstart scenario, after an underrun
1297 ; we return true, since we need more data
1298 ;
1299 ; case #3: curbuf is empty, nextbuf is empty, not-usedbuf is passed
1300 ; -- happens after an underrun (2nd step) or something like that, where we
1301 ; missed the polling loop
1302 ; we return false, after placing the new data in and starting DMA
1303 ;
1304 ; yes:
1305 ; case #4: curbuf is occupied, nextbuf empty, curbuf passed
1306 ; return true -- we've used data and have space for more
1307 ; but, don't do anything
1308 ;
1309 ; case #5: curbuf is occupied, nextbuf is empty, somebuf is passed
1310 ; return false -- we are accepting new data
1311 ; we place somebuf into curbuf, and let DMA continue
1312 ;
1313 ;
1314 ; Arguments:      a1 - buffer passed, may be new data or old
1315 ;                  b1 - length of this passed buffer
1316 ;
1317 ; Return Values:  None.
1318 ;
1319 ; Global Variables: None.
1320 ;
1321 ; Shared Variables: None.
1322 ;
1323 ; Local Variables: None.
1324 ;
1325 ; Inputs:         None.
1326 ;
1327 ; Outputs:        None.
1328 ;
1329 ; Error Handling:  None.
1330 ;
1331 ; Algorithm:      None.
1332 ;
1333 ; Data Structures: None.
1334 ;
1335 ; Registers changed: None.
1336 ;
1337 ; Limitations:    None known.
1338 ;
1339 ; Known Bugs:     None.
1340 ;
1341 ; Special Notes:
1342 ;
1343 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
1344 ;
1345     global Fupdate

```

```

1346 Fupdate:                                ;C entry point
1347
1348     ;standard C prologue
1349     move    #0,n6                        ;k is the amount of local space
1350     move    ssh,x:(r6)+                  ;save the return address
1351     move    (r6)+n6                      ;allocate local stack space of size k
1352
1353
1354     ;save all registers we use so we can restore later
1355     PUSH_ACC b
1356     move    x0,ssh
1357     move    x1,ssl
1358
1359     move    r0,ssh
1360     move    r1,ssl
1361
1362     ;we can only cmp to x/y and accumulators, so let's
1363     ;move the arguments given into x0 and x1:
1364     move    a1,x0                        ;move buffer address into x0
1365     move    a1,r0
1366     move    b1,x1                        ;move buffer length into x1
1367     move    b1,r1
1368
1369     START_CRITICAL
1370     ;we check if both curbuf and nextbuf are occupied
1371     move    x:nextbuflen,b
1372     cmp     #>NO_BUFFER_LEN,b
1373     jeq     update_empty_nextbuf        ;if nextbuf is empty, we have more
                                         options
1374
1375     update_full_buffers:
1376     ;scenario #1: full nextbuf, curbuf
1377     ;but, if the buffer doesn't fit, we must acquit
1378     ; (we have no space to handle new data)
1379     move    #>FALSE,a                    ;set our return value
1380     jmp     update_end
1381
1382     update_empty_nextbuf:
1383     ;there are four possible scenarios here. we check occupancy of curbuf
1384     ; to decide our route
1385     move    x:curbuflen,b
1386     cmp     #>NO_BUFFER_LEN,b
1387     jne     update_occupied_curbuf      ;we have something in curbuf
1388
1389     ;scenario #2/3: empty curbuf
1390     ; this is a TX underrun, so we get these two scenarios out of the way
1391
1392     ;to distinguish between #2 & #3, we need to check passed buf with
     usedbuf
1393     update_empty_curbuf:
1394     move    x:usedbuf,b
1395     cmp     x0,b                        ;check if we've used this buffer already
1396     jne     update_empty_curbuf_new
1397
1398     ;update_empty_curbuf_used: ;scenario #2: we have empty buffers, and this
1399     ;buffer's old
1400     move    #>TRUE,a

```

```

1401         jmp      update_end
1402
1403 update_empty_curbuf_new: ;scenario #3: we have all empty buffers, and new data
1404         move     x0,x:curbuf      ;place in the new buffer
1405         move     x1,x:curbuflen
1406         jsr      process_mp3_buffer ;and process the buffer when we place it in
1407                                   ;(r0-r1 set up top)
1408
1409         move     x0,x:M_DSR2      ;also initialize DMA source/count
1410         move     x1,b             ;we need to account for DMA transfers = DCO + 1
1411         sub      #>1,b
1412         move     b1,x:M_DC02
1413
1414         ;we now need to activate DMA to act on this new data
1415         bset     #M_DE,x:M_DCR2 ;activate
1416
1417         move     #>FALSE,a
1418         jmp      update_end
1419
1420         ;scenario #4/5: we have current data, but could accept more
1421         ; to differentiate the two, we need to compare passed to curbuf
1422 update_occupied_curbuf:
1423         move     x:curbuf,b
1424         cmp      x0,b
1425         jne      update_occupied_curbuf_new
1426
1427 update_occupied_curbuf_old: ;scenario #4: we have data in curbuf, and passed
old data
1428         move     #>TRUE,a        ;we need more new, not old data
1429         jmp      update_end
1430
1431 update_occupied_curbuf_new: ;scenario #5: we have new data in passed buf.
1432         ;process our data first:
1433         jsr      process_mp3_buffer
1434         move     x0,x:nextbuf     ;we change the next pointer to reflect new data
1435         move     x1,x:nextbuflen
1436         ;DMA magic handles the rest
1437
1438         move     #>FALSE,a
1439         jmp      update_end
1440
1441
1442
1443 update_end:
1444
1445     END_CRITICAL
1446
1447 ;end of body code, now restore all registers
1448         move     ssl,r1
1449         move     ssh,r0
1450
1451
1452         move     ssl,x1
1453         move     ssh,x0
1454         POP_ACC  b
1455
1456

```

```

1457         ;standard C epilogue
1458         move    #0+1,n6
1459         move    (r6)-n6        ;deallocate local stack space, set ccr flags
1460         tst     a
1461         move    x:(r6),ssh     ;get return address
1462
1463         nop      ;pipeline cache issue (see errata)
1464         rts      ;return from subroutine
1465
1466 ;end Fupdate
1467
1468 ; process_mp3_buffer
1469 ; Functional Specification
1470 ;
1471 ; Description:      process_mp3_buffer processes each byte of the
1472 ; mp3 buffer given to it. it assumes that the data is in the lower
1473 ; 16 bits of the words given to it, and that we send to ESSi via
1474 ; DMA.
1475 ;
1476 ; Operation:       we copy the word into a1 and b1, and left-align the
1477 ; lower 8 bits in a1 (16 bit shift left). we then clear the top 8 bits of
1478 ; b1, and or in b1 with x1 in order to successfully swap order. this is
1479 ; then written back out.
1480 ;
1481 ; Arguments:       r0 - pointer to mp3 buffer
1482 ;                  r1 - length of mp3 buffer
1483 ;
1484 ; Return Values:   None.
1485 ;
1486 ; Global Variables: None.
1487 ;
1488 ; Shared Variables: None.
1489 ;
1490 ; Local Variables: None.
1491 ;
1492 ; Inputs:          None.
1493 ;
1494 ; Outputs:         None.
1495 ;
1496 ; Error Handling:  None.
1497 ;
1498 ; Algorithm:       None.
1499 ;
1500 ; Data Structures: None.
1501 ;
1502 ; Registers changed: None.
1503 ;
1504 ; Limitations:     None known.
1505 ;
1506 ; Known Bugs:      None.
1507 ;
1508 ; Special Notes:
1509 ;
1510 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
1511 ;
1512         global  process_mp3_buffer
1513 process_mp3_buffer:                ;assembly entry point

```

```

1514
1515     ;save all registers we use so we can restore later
1516     PUSH_ACC a
1517     PUSH_ACC b
1518     move r0,ssh
1519     move x0,ssl
1520
1521     nop
1522     do r1,process_buffer_loop
1523     nop
1524
1525     move x:(r0),a ;we copy into a/b so we can do bitshifts.
1526     nop
1527     move a1,b
1528
1529     lsl #16,a ;left-align the lower byte
1530     and #>$ff00,b ;select the top byte from b
1531     move b1,x0 ;move into x0 so that we can use or b1 with a
1532     or x0,a ;we find our fixed word
1533
1534     move a1,x:(r0)+ ;move our word back in
1535
1536     nop ;spacing nops
1537     nop
1538     nop
1539 process_buffer_loop:
1540
1541
1542     ;end of body code, now restore all registers
1543     move ssl,x0
1544     move ssh,r0
1545     POP_ACC b
1546     POP_ACC a
1547     nop
1548     rts ;return from subroutine
1549 ;end function
1550
1551
1552
1553
1554 ; Fsetup_dac
1555 ; Functional Specification
1556 ;
1557 ; Description:      Fsetup_dac sets up the ESS11 registers for the DAC's
control
1558 ; interface, and sets it to reasonable values:
1559 ;   - 192KHz oversampling
1560 ;   - analog mute/infinite zero detect
1561 ;   - auto-ramp volume
1562 ;   - anti-clipping mode
1563 ;   - filter #2: minimum phase linear (like AMB's gamma-2 setting B)
1564 ;
1565 ; Operation:      Fsetup_dac writes the ESS1 registers, then calls
1566 ; write_sync_dac to commit the commands.
1567 ;
1568 ; Arguments:      None.
1569 ;

```

```

1570 ; Return Values:    None.
1571 ;
1572 ; Global Variables:  None.
1573 ;
1574 ; Shared Variables:  None.
1575 ;
1576 ; Local Variables:   None.
1577 ;
1578 ; Inputs:            None.
1579 ;
1580 ; Outputs:           None.
1581 ;
1582 ; Error Handling:    None.
1583 ;
1584 ; Algorithm:         None.
1585 ;
1586 ; Data Structures:   None.
1587 ;
1588 ; Registers changed: None.
1589 ;
1590 ; Limitations:       None known.
1591 ;
1592 ; Known Bugs:       None.
1593 ;
1594 ; Special Notes:     None.
1595 ;
1596 ; Revision History:  2011/3/25    Raymond Jimenez    first revision
1597 ;
1598     global  Fsetup_dac
1599 Fsetup_dac:                ;C entry point
1600
1601     ;standard C prologue
1602     move    #0,n6          ;k is the amount of local space
1603     move    ssh,x:(r6)+    ;save the return address
1604     move    (r6)+n6        ;allocate local stack space of size k
1605
1606
1607     ;save all registers we use so we can restore later
1608     move    r0,ssh
1609     move    r1,ssl
1610
1611     ;initialize the ESS1 1 control registers
1612     move    #>ESS11_CRA,r0
1613     move    r0,x:M_CRA1
1614     move    #>ESS11_CRB,r0
1615     move    r0,x:M_CRB1
1616
1617     ;now configure the ESS1 1 pins as active and not GPIO
1618     move    #>ESS11_PCRD,r0
1619     move    r0,x:M_PCRD
1620     move    #>ESS11_PRRD,r0
1621     move    r0,x:M_PRRD
1622
1623     ;we load first data
1624     move    #>$0,r0
1625     move    r0,x:M_TX10
1626

```

```

1627     ;enable transmitter
1628     bset     #M_SSTE0,x:M_CRB1
1629
1630
1631     ;now that the ESSI port is setup, we can go ahead and write our initial
1632
1632     ; reg values
1633     move     #>(DAC_RESET_ADDR),r0
1634     jsr      write_sync_dac           ;reset the chip to a known state
1635
1636
1637     move     #>(DAC_VOL_ADDR|DAC_VOL_CONTROL),r0      ;setup vol control
1638     jsr      write_sync_dac
1639     move     #>(DAC_FILTER_ADDR|DAC_FILTER_CONTROL),r0
1640                                           ;setup filters
1641     jsr      write_sync_dac
1642     move     #>(DAC_MODE1_ADDR|DAC_MODE_CONTROL_1),r0
1643                                           ;setup oversampling
1644     jsr      write_sync_dac
1645
1646     move     #>INIT_ATTEN,r0
1647     move     r0,x:sound_atten
1648
1649     ;set our initial volume
1650     jsr      Fvol_down
1651     jsr      Fvol_up
1652
1653     ;end of body code, now restore all registers
1654     move     ssl,r1
1655     move     ssh,r0
1656
1657     ;standard C epilogue
1658     move     #0+1,n6
1659     move     (r6)-n6           ;deallocate local stack space, set ccr flags
1660     tst      a
1661     move     x:(r6),ssh       ;get return address
1662
1663     nop      ;pipeline cache issue (see errata)
1664     rts      ;return from subroutine
1665
1666     ;end Fsetup_dac
1667
1668
1669     ; write_sync_dac
1670     ; Functional Specification
1671     ;
1672     ; Description:      write_sync_dac writes a DAC register value to the
1673     ; DAC.
1674     ;
1675     ; ESSI1 interrupts must be disabled before invocation; they should never
1676     ; be enabled in the first place.
1677     ;
1678     ; Operation:      write_sync_dac simply waits on TDE to ensure that all
1679     ; data has been transmitted. it then moves the requested data and transmits
1680     ; it, and then returns. the magic of ESSI takes care of our frame sync.
1681     ;
1682     ; Arguments:      r0 - data to write, left aligned

```

```

1683 ;               r0[23:17] - address of register to write
1684 ;               r0[16] - should always be 0
1685 ;               r0[15-8] - data to write
1686 ;               r0[7-0] - reserved; write 0 for compat
1687 ;
1688 ; Return Values:   None.
1689 ;
1690 ; Global Variables: None.
1691 ;
1692 ; Shared Variables: None.
1693 ;
1694 ; Local Variables: None.
1695 ;
1696 ; Inputs:          None.
1697 ;
1698 ; Outputs:          None.
1699 ;
1700 ; Error Handling:   None.
1701 ;
1702 ; Algorithm:        None.
1703 ;
1704 ; Data Structures: None.
1705 ;
1706 ; Registers changed: None.
1707 ;
1708 ; Limitations:      None known.
1709 ;
1710 ; Known Bugs:       None.
1711 ;
1712 ; Special Notes:    We require the strange bitfield structure of r0 to make
1713 ; things faster; since we mostly deal with pre-composed constants, we can use
1714 ; macro shifts to pre-calculate everything.
1715 ;
1716 ; Revision History: 2011/03/25   Raymond Jimenez   first revision
1717 ;
1718     global write_sync_dac
1719 write_sync_dac:                ;assembly entry point
1720
1721     ;save all registers we use so we can restore later
1722
1723     ;we first make sure the transmitter is empty
1724     jclr    #M_TDE,x:M_SSISR1,*    ;keep on jumping here until TDE is set
1725     move    r0,x:M_TX10            ;transmit first 16 bits: command, address
1726
1727     nop    ;respect the 3-cycle pipeline delay; we return when flags have
        been set
1728         nop
1729         nop
1730
1731     ;restore all registers. we have.. none?
1732
1733     nop    ;pipeline cache issue (see errata)
1734     rts    ;return from subroutine
1735 ;end write_sync_dac
1736
1737 ; Fvol_up
1738 ; Functional Specification

```



```

1739 ;
1740 ; Description:      vol_up is called whenever we want to increase the volume.
1741 ; it
1742 ; increases it by ten -.125dB steps.
1743 ; Operation:       vol_up simply calls write_sync_dac in order to tell the dac
1744 ; go up to the desired volume.
1745 ;
1746 ; Arguments:       None.
1747 ;
1748 ; Return Values:   None.
1749 ;
1750 ; Global Variables: None.
1751 ;
1752 ; Shared Variables: None.
1753 ;
1754 ; Local Variables: None.
1755 ;
1756 ; Inputs:          None.
1757 ;
1758 ; Outputs:         None.
1759 ;
1760 ; Error Handling:   None.
1761 ;
1762 ; Algorithm:        None.
1763 ;
1764 ; Data Structures: None.
1765 ;
1766 ; Registers changed: None.
1767 ;
1768 ; Limitations:      None known.
1769 ;
1770 ; Known Bugs:       None.
1771 ;
1772 ; Special Notes:
1773 ;
1774 ; Revision History: 2011/03/27      Raymond Jimenez      first revision
1775 ;
1776     global  Fvol_up
1777 Fvol_up:      ;C entry point
1778
1779     ;standard C prologue
1780     move     #0,n6          ;k is the amount of local space
1781     move     ssh,x:(r6)+    ;save the return address
1782     move     (r6)+n6        ;allocate local stack space of size k
1783
1784
1785     ;save all registers we use so we can restore later
1786     PUSH_ACC a
1787     move     x0,ssh
1788     move     r0,ssl
1789
1790
1791     move     x:sound_atten,a      ;we grab our current attenuation state
1792     move     #>ATTEN_STEP,x0     ;move step into x0 so we can subtract
1793     it

```

```

1793     sub     x0,a                ; subtract (smaller atten == louder)
1794
1795     cmp     #>MIN_ATTEN,a        ;see if it's too loud, nowhere to go
1796     jlt     vol_up_done          ;too loud, so go don't make any changes
1797
1798
1799     move     a1,x:sound_atten     ;store the change
1800
1801     and     #>ATTEN_MASK,a        ;we now have our desired attenuation
1802                                     ; in a1, so we use masks to get it to
1803                                     ; the DAC.
1804
1805     move     a1,x0                ;keep a copy in x0, we'll be needing it
1806                                     ;(we do two 5-bit writes)
1807     and     #>ATTEN_REG_MASK,a    ;clear out to leave the bottom 5 bits
1808
1809     ;we now transmit the lower 5 bits
1810     lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
1811     or      #>DAC_LLSB_ADDR,a      ;compose address
1812
1813     move     a1,r0                ;ready for calling
1814     jsr     write_sync_dac         ;effect the change
1815
1816     ;now we transmit the high 5 bits, and latch the changes
1817     move     x0,a                 ;grab the old copy
1818     lsr     #5,a                  ;dac volume register is 5 bits wide per
1819     and     #>ATTEN_REG_MASK,a      ;clear out the top 5 bits
1820     lsl     #DAC_DATA_SHIFT,a      ;shift as necessary for write_sync_dac
1821     or      #>(DAC_LMSB_ADDR | (ATTEN_LATCH_MASK<<DAC_DATA_SHIFT)),a
1822                                     ;compose address & commit change bit
1823     move     a1,r0
1824     jsr     write_sync_dac         ;really effect the change
1825
1826 vol_up_done:
1827
1828 ;end of body code, now restore all registers
1829     move     ssl,r0
1830     move     ssh,x0
1831     POP_ACC a
1832
1833
1834     ;standard C epilogue
1835     move     #0+1,n6
1836     move     (r6)-n6              ;deallocate local stack space, set ccr flags
1837     tst     a
1838     move     x:(r6),ssh          ;get return address
1839
1840     nop     ;pipeline cache issue (see errata)
1841     rts     ;return from subroutine
1842
1843 ;end Fvol_up
1844
1845
1846 ; Fvol_down
1847 ; Functional Specification
1848 ;
1849 ; Description:      vol_down is called whenever we want to increase the volume.

```

```

1849 it
1850 ; increases it by ten -.125dB steps.
1851 ;
1852 ; Operation:      vol_down simply calls write_sync_dac in order to tell the
                    dac
1853 ; go up to the desired volume.
1854 ;
1855 ; Arguments:      None.
1856 ;
1857 ; Return Values:  None.
1858 ;
1859 ; Global Variables: None.
1860 ;
1861 ; Shared Variables: None.
1862 ;
1863 ; Local Variables: None.
1864 ;
1865 ; Inputs:         None.
1866 ;
1867 ; Outputs:        None.
1868 ;
1869 ; Error Handling:  None.
1870 ;
1871 ; Algorithm:       None.
1872 ;
1873 ; Data Structures: None.
1874 ;
1875 ; Registers changed: None.
1876 ;
1877 ; Limitations:     None known.
1878 ;
1879 ; Known Bugs:      None.
1880 ;
1881 ; Special Notes:
1882 ;
1883 ; Revision History: 2011/03/27    Raymond Jimenez    first revision
1884 ;
1885     global  Fvol_down
1886 Fvol_down:                ;C entry point
1887
1888     ;standard C prologue
1889     move    #0,n6          ;k is the amount of local space
1890     move    ssh,x:(r6)+    ;save the return address
1891     move    (r6)+n6        ;allocate local stack space of size k
1892
1893
1894     ;save all registers we use so we can restore later
1895     PUSH_ACC a
1896     move    x0,ssh
1897     move    r0,ssl
1898
1899
1900     move    x:sound_atten,a    ;we grab our current attenuation state
1901     move    #>ATTEN_STEP,x0    ;move step into x0 so we can subtract
1902     it
1903     add     x0,a                ; add (more atten == softer)

```

```

1904      cmp     #>MAX_ATTEN,a           ; see if it's too soft.
1905      jgt     vol_down_done           ;too soft, so go don't make any
                                         changes
1906
1907
1908      move     a1,x:sound_atten        ;store the change
1909      and      #>ATTEN_MASK,a          ;we now have our desired attenuation
1910                                         ; in a1, so we use masks to get it to
1911                                         ; the DAC.
1912
1913      move     a1,x0                    ;keep a copy in x0, we'll be needing it
1914                                         ;(we do two 5-bit writes)
1915      and      #>ATTEN_REG_MASK,a      ;clear out the top 5 bits
1916
1917      ;we now transmit the lower 5 bits
1918      lsl      #DAC_DATA_SHIFT,a       ;shift as necessary for write_sync_dac
1919      or       #>DAC_LLSB_ADDR,a       ;compose address
1920
1921      move     a1,r0                    ;ready for calling
1922      jsr      write_sync_dac           ;effect the change
1923
1924      ;now we transmit the high 5 bits, and latch the changes
1925      move     x0,a                     ;grab the old copy
1926      lsr      #5,a                     ;dac volume register is 5 bits wide per
1927      and      #>ATTEN_REG_MASK,a      ;clear out the top 5 bits
1928      lsl      #DAC_DATA_SHIFT,a       ;shift as necessary for write_sync_dac
1929      or       #>(DAC_LMSB_ADDR|(ATTEN_LATCH_MASK<<DAC_DATA_SHIFT)),a
1930                                         ;compose address & commit change bit
1931      move     a1,r0
1932      jsr      write_sync_dac           ;really effect the change
1933
1934      vol_down_done:
1935
1936      ;end of body code, now restore all registers
1937      move     ssl,r0
1938      move     ssh,x0
1939      POP_ACC a
1940
1941
1942      ;standard C epilogue
1943      move     #0+1,n6
1944      move     (r6)-n6                  ;deallocate local stack space, set ccr flags
1945      tst      a
1946      move     x:(r6),ssh               ;get return address
1947      nop      ;pipeline cache issue (see errata)
1948      rts      ;return from subroutine
1949
1950      ;end Fvol_down
1951
1952
1953
1954
1955      ; get_atten
1956      ; Functional Specification
1957      ;
1958      ; Description:      This function returns the current attenuation
1959      ; status in steps of .125dB.

```

```
1960 ;
1961 ; Operation:      This function is an accessor function for sound_atten;
1962 ; it moves sound_atten into a.
1963 ;
1964 ; Arguments:      None.
1965 ;
1966 ; Return Values:   a1 - current attenuation in steps of .125dB
1967 ;
1968 ; Global Variables: None.
1969 ;
1970 ; Shared Variables: None.
1971 ;
1972 ; Local Variables: None.
1973 ;
1974 ; Inputs:         None.
1975 ;
1976 ; Outputs:        None.
1977 ;
1978 ; Error Handling:  None.
1979 ;
1980 ; Algorithm:       None.
1981 ;
1982 ; Data Structures: None.
1983 ;
1984 ; Registers changed: None.
1985 ;
1986 ; Limitations:     None known.
1987 ;
1988 ; Known Bugs:      None.
1989 ;
1990 ; Special Notes:
1991 ;
1992 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
1993 ;
1994     global  get_atten
1995 get_atten:                ;assembly entry point
1996
1997     ;save all registers we use so we can restore later
1998
1999     move    x:sound_atten,a
2000
2001     ;end of body code, now restore all registers
2002     rts     ;return from subroutine
2003 ;end get_atten
2004
2005
2006     endsec
2007
2008
2009
```

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.

```
1  TICKS_PER_MS      EQU      1
2  SLEEP_CYCLES      EQU      8
3
```

```

1  ;
2  ; timing.asm
3  ;
4  ; This file offers several timing provider routines.
5  ;
6
7      section timing
8  INCLUDE 'timing.inc'
9  INCLUDE 'macros.inc'
10     org      x:
11
12     global time_elapsed_counter
13 time_elapsed_counter:
14     ds        $1                ;one word to count the clock ticks elapsed.
15
16     org      p:                ;we want to be located in program space
17
18 ; Felapsed_time
19 ; Functional Specification
20 ;
21 ; Description:      Felapsed_time provides the number of milliseconds it has
22 ; been since it was last called.
23 ;
24 ; Operation:        It retrieves the value of the time_elapsed_counter, divides
25 ; it by TICKS_PER_MS, and then returns this value.
26 ;
27 ; Arguments:         None.
28 ;
29 ; Return Values:     None.
30 ;
31 ; Global Variables:  None.
32 ;
33 ; Shared Variables:  None.
34 ;
35 ; Local Variables:   None.
36 ;
37 ; Inputs:            None.
38 ;
39 ; Outputs:           None.
40 ;
41 ; Error Handling:    None.
42 ;
43 ; Algorithm:         None.
44 ;
45 ; Data Structures:   None.
46 ;
47 ; Registers changed: None.
48 ;
49 ; Limitations:       None known.
50 ;
51 ; Known Bugs:        None.
52 ;
53 ; Special Notes:
54 ;
55 ; Revision History:  2011/02/16      Raymond Jimenez      first revision
56 ;
57     global Felapsed_time

```



```

58 Felapsed_time:           ;C entry point
59     ;standard C prologue
60     move    #0,n6         ;k is the amount of local space
61     move    ssh,x:(r6)+    ;save the return address
62     move    (r6)+n6       ;allocate local stack space of size k
63
64     ;save all registers we use so we can restore later
65     move    r0,ssh
66     move    x0,ssl
67
68     move    #>0,r0
69
70     ;we turn off interrupts for a bit to ensure we don't lose ticks
71     ; there is no easy atomic alternative (no XCHG even!)
72     START_CRITICAL
73     move    x:time_elapsed_counter,a
74     move    r0,x:time_elapsed_counter
75     END_CRITICAL
76
77     ;we rely on our handler being called every millisecond.
78
79     ;end of body code, now restore all registers
80     move    ssl,x0
81     move    ssh,r0
82
83     ;standard C epilogue
84     move    #0+1,n6
85     move    (r6)-n6        ;deallocate local stack space, set ccr flags
86     tst     a
87     move    x:(r6),ssh     ;get return address
88     rts     ;return from subroutine
89 ;end Felapsed_time
90
91 ; time_elapsed_update
92 ; Functional Specification
93 ;
94 ; Description:      time_elapsed_update should be called every
95 ; millisecond. It updates a counter in memory, and it is the job
96 ; of time_elapsed to properly interpret this counter's value.
97 ;
98 ; Operation:       It pulls in the counter into a register, increments
99 ; it, and writes it back out to memory.
100 ;
101 ; Arguments:       None.
102 ;
103 ; Return Values:   None.
104 ;
105 ; Global Variables: None.
106 ;
107 ; Shared Variables: None.
108 ;
109 ; Local Variables: None.
110 ;
111 ; Inputs:         None.
112 ;
113 ; Outputs:        None.
114 ;

```

```

115 ; Error Handling:   None.
116 ;
117 ; Algorithm:        None.
118 ;
119 ; Data Structures:  None.
120 ;
121 ; Registers changed: None.
122 ;
123 ; Limitations:      None known.
124 ;
125 ; Known Bugs:       None.
126 ;
127 ; Special Notes:
128 ;
129 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
130 ;
131     global time_elapsed_update
132 time_elapsed_update:                ;assembly entry point
133
134     ;save all registers we use so we can restore later
135     move    r0,ssh                  ;we save r0 since we alter it
136
137
138     move    x:time_elapsed_counter,r0 ;pull in our current counter
139     move    (r0)+                      ;increment it
140     move    r0,x:time_elapsed_counter ;then move it back
141
142
143     ;end of body code, now restore all registers
144     move    ssh,r0                  ;and restore r0 to original state.
145
146     rts    ;return from subroutine
147 ;end time_elapsed_update
148
149 ; sleep
150 ; Functional Specification
151 ;
152 ; Description:       sleep implements a sleep function using a delay loop.
153 ; It is guaranteed to give a minimum of the sleep period, but the maximum
154 ; is undefined (due to interrupts it is possible to sleep forever, beware.)
155 ;
156 ; Operation:         sleep simply loops the specified number of times, reading
157 ; from an SRAM memory address to ensure the DSP can't optimize the loop away.
158 ;
159 ; Arguments:         r0 - time to sleep in multiples of 100ns
160 ;
161 ; Return Values:     None.
162 ;
163 ; Global Variables:  None.
164 ;
165 ; Shared Variables:  None.
166 ;
167 ; Local Variables:   None.
168 ;
169 ; Inputs:            None.
170 ;
171 ; Outputs:           None.

```

```

172 ;
173 ; Error Handling:   None.
174 ;
175 ; Algorithm:        None.
176 ;
177 ; Data Structures:  None.
178 ;
179 ; Registers changed: None.
180 ;
181 ; Limitations:      None known.
182 ;
183 ; Known Bugs:       None.
184 ;
185 ; Special Notes:
186 ;
187 ; Revision History: 2011/02/16   Raymond Jimenez   first revision
188 ;
189     global  sleep
190 sleep:                                ;assembly entry point
191
192     ;save all registers we use so we can restore later
193     move    r1,ssh
194
195
196     do      r0,outer_sleep_loop
197
198     do      #SLEEP_CYCLES,inner_sleep_loop
199
200     move    x:$0,r1                ;this instruction ensures at least one clock
201                                     ;period of wait, due to internal SRAM access
202
203     ;spacing nops
204     nop
205     nop
206     nop
207 inner_sleep_loop:
208
209     ;spacing nops
210     nop
211     nop
212     nop
213 outer_sleep_loop:
214
215
216     ;end of body code, now restore all registers
217     move    ssh,r1
218
219     rts     ;return from subroutine
220 ;end sleep
221
222
223     endsec
224
225
226

```

8.9 Interrupts

`intrrpts.asm` contains the interrupt installation/initialization code.

```

1  ; interrupts.asm
2  ;
3  ; This file offers interrupt setup and handler routines.
4  ;
5  ; void setup_timer0(void) -- sets up timer 0, currently used for processing
6  ;   display commands at 500ns intervals.
7  ;
8  ; void install_timer0_handler(void) -- installs timer0 handler (currently calls
9  ;   display function only)
10 ;
11 ; void setup_keypad_irq(void) -- sets up IRQ A, the keypad interrupt, and
    installs
12 ;   the keypad handler
13 ;
14
15         section interrupts
16 INCLUDE 'DSP_EQU.inc'
17 INCLUDE 'display.inc'
18 INCLUDE 'macros.inc'
19         org      x:
20
21
22         org      p:           ;we want to be located in program space
23
24 ; Fstart_intr_setup
25 ; Functional Specification
26 ;
27 ; Description:      Fstart_intr_setup should be called before setting
28 ; up any interrupts. (it disables them).
29 ;
30 ; Operation:       It forces the SR interrupt mask pins to be 3, masking
31 ; all interrupts.
32 ;
33 ; Arguments:       None.
34 ;
35 ; Return Values:   None.
36 ;
37 ; Global Variables: None.
38 ;
39 ; Shared Variables: None.
40 ;
41 ; Local Variables: None.
42 ;
43 ; Inputs:          None.
44 ;
45 ; Outputs:         None.
46 ;
47 ; Error Handling:  None.
48 ;
49 ; Algorithm:       None.
50 ;
51 ; Data Structures: None.
52 ;
53 ; Registers changed: None.
54 ;
55 ; Limitations:     None known.
56 ;

```

```

57 ; Known Bugs:      None.
58 ;
59 ; Special Notes:
60 ;
61 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
62 ;
63     global  Fstart_intr_setup
64 Fstart_intr_setup:      ;C entry point
65     ;standard C prologue
66     move    #0,n6        ;k is the amount of local space
67     move    ssh,x:(r6)+    ;save the return address
68     move    (r6)+n6        ;allocate local stack space of size k
69
70     ;save all registers we use so we can restore later (we don't use any)
71
72     FORCE_INTERRUPTS_OFF
73
74     ;end of body code, now restore all registers
75
76     ;standard C epilogue
77     move    #0+1,n6
78     move    (r6)-n6        ;deallocate local stack space, set ccr flags
79     tst     a
80     move    x:(r6),ssh    ;get return address
81     nop
82     rts        ;return from subroutine
83 ;end Fstart_intr_setup
84
85
86     global  Fsetup_timer0    ;prologue
87 Fsetup_timer0:      ;C entry point
88
89     move    #0,n6        ;k is the amount of local space
90     move    ssh,x:(r6)+    ;save the return address
91     move    (r6)+n6        ;allocate local stack space of size k
92
93 ;Save all registers to be used.
94     ;put a on stack, must keep it clean
95     PUSH_ACC a
96     move    x0,ssh
97
98
99 ;Function body.
100
101     ;we setup the interrupt handler before enabling the timer
102     move    #>$0bf080,x0    ;unconditional branch to our subroutine
103                               ;(long interrupt)
104     move    x0,p:I_TIM0C    ;install handler
105     move    #>timer0_handler,x0
106     move    x0,p:I_TIM0C+1    ;install address of our handler
107
108     ;and remember to enable the interrupt handler globally
109     ;(SR interrupt bits are already set by crt0)
110     movep   x:M_IPRP,a
111     and     #~M_T0L,a        ;clear the timer bits of the IPRP
112     or      #(2<<M_T0L0),a    ;and force it to high (but
113                               ; maskable) priority

```

```

114         movep    a1,x:M_IPRP                ;write back to the register
115
116         ;setup the prescaler timer to have 4 cycles.
117         ;@ 40MHz = 80/2 MHz, this gives us 1us resolution
118         movep    #40,x:M_TPLR
119
120         ;we want to have 200 cycles -> 200us command cycle time for the display
121         movep    #200,x:M_TCPR0
122         ;initialize the timer start count to zero, too
123         movep    #0,x:M_TLR0
124
125         ;set timer 0 to be enabled and in event counting mode, continuously
126         ; counting and set it to trigger interrupts.
127         movep    #(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR0
128
129         ;then set the timer to enable.
130         movep    #(1<<M_TE)|(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR0
131
132
133         ;Restore all registers used.
134         move     ssh,x0
135         POP_ACC  a
136
137         move     #0+1,n6
138         move     (r6)-n6                ;deallocate local stack space, set ccr flags
139         tst      a
140         move     x:(r6),ssh             ;get return address
141         nop
142         rts
143
144
145         global   Fsetup_timer1          ;prologue
146         Fsetup_timer1:                  ;C entry point
147
148         move     #0,n6                  ;k is the amount of local space
149         move     ssh,x:(r6)+             ;save the return address
150         move     (r6)+n6                 ;allocate local stack space of size k
151
152         ;Save all registers to be used.
153         ;put a on stack, must keep it clean
154         PUSH_ACC a
155         move     x0,ssh
156
157
158         ;Function body.
159
160         ;we setup the interrupt handler before enabling the timer
161         move     #>$0bf080,x0           ;unconditional branch to our subroutine
162                                         ;(long interrupt)
163         move     x0,p:I_TIM1C           ;install handler
164         move     #>timer1_handler,x0
165         move     x0,p:I_TIM1C+1         ;install address of our handler
166
167         ;and remember to enable the interrupt handler globally
168         ;(SR interrupt bits are already set by crt0)
169         ; this should have been setup already by timer0_setup.
170

```

```

171         ; 1ms per tick.
172         ; additionally, we base our MP3 timing off of this routine,
173         ; down to millisecond accuracy, so don't change!
174         movep    #1000,x:M_TCPR1
175         ;initialize the timer start count to zero, too
176         movep    #0,x:M_TLR1
177
178         ;set timer 1 to be enabled and in event counting mode, continuously
179         ; counting and set it to trigger interrupts.
180         movep    #(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR1
181
182         ;then set the timer to enable.
183         movep    #(1<<M_TE)|(1<<M_TCIE)|3|(1<<M_PCE)|(1<<M_TRM),x:M_TCSR1
184
185
186         ;we also setup the time_elapsed_counter which time_elapsed depends
187         ; on:
188         move     #>0,x0
189         move     x0,x:time_elapsed_counter
190
191     ;Restore all registers used.
192         move     ssh,x0
193         POP_ACC  a
194
195         move     #0+1,n6
196         move     (r6)-n6             ;deallocate local stack space, set ccr flags
197         tst      a
198         move     x:(r6),ssh         ;get return address
199         nop
200         rts
201
202
203
204
205     global  Fsetup_timer2           ;prologue
206 Fsetup_timer2:                     ;C entry point
207
208         move     #0,n6             ;k is the amount of local space
209         move     ssh,x:(r6)+         ;save the return address
210         move     (r6)+n6             ;allocate local stack space of size k
211
212     ;Save all registers to be used.
213         ;put a on stack, must keep it clean
214         PUSH_ACC a
215         move     x0,ssh
216
217
218     ;Function body.
219
220         ;we setup the interrupt handler before enabling the timer
221         move     #>$0bf080,x0       ;unconditional branch to our subroutine
222                                     ;(long interrupt)
223         move     x0,p:I_TIM2C       ;install handler
224         move     #>timer2_handler,x0
225         move     x0,p:I_TIM2C+1     ;install address of our handler
226
227         ;and remember to enable the interrupt handler globally

```



```

228      ;(SR interrupt bits are already set by crt0)
229      ; this should have been setup already by timer0_setup.
230
231      ; 15us per tick.
232      ; dram refresh, so it's important that this works properly.
233      movep    #400,x:M_TCPR2
234      ;initialize the timer start count to zero, too
235      movep    #0,x:M_TLR2
236
237      ;set timer 2 to be enabled and in GPIO mode, continuously
238      ; counting and set it to trigger interrupts. we latch off the
239      ; internal clock/2 so that we have a 25ns pulse instead of some long
240      ; thing.
241      movep    #(1<<M_TCIE)|(0<<4)|(0<<M_PCE)|(1<<M_TRM)|(1<<M_DIR),x:M_TCSR2
242
243      ;then set the timer to enable.
244      movep    #(1<<M_TE)|(1<<M_TCIE)|(0<<4)|(0<<M_PCE)|(1<<M_TRM)|(1<<M_DIR),x:M_TCSR2
245
246      ;Restore all registers used.
247      move     ssh,x0
248      POP_ACC  a
249
250      move     #0+1,n6
251      move     (r6)-n6      ;deallocate local stack space, set ccr flags
252      tst      a
253      move     x:(r6),ssh   ;get return address
254      nop
255      rts
256
257
258
259
260
261      global  timer0_handler ;prologue
262 timer0_handler:      ;interrupt entry point
263
264      jsr      display_update ;call display update function
265
266      rti      ;return; interrupts return with RTI
267
268
269
270      global  timer1_handler ;prologue
271
272 timer1_handler:      ;interrupt entry point
273
274      jsr      time_elapsed_update ;call time_elapsed update function
275
276      rti      ;return; interrupts return with RTI
277
278
279 timer2_handler:      ;interrupt entry point
280
281      ;how long does a refresh take? well, let's see, we need to hold the bus
282      ; for about... 300ns? we'll hold for 400 just in case.
283      ;each instruction is ~12.5ns, so that's 32 instructions.

```

```

284
285
286
287     ; save registers we use
288     move    r0,ssh
289     move    r1,ssl
290     move    r2,ssh
291     move    #$200000,r0      ;we just grab the first 32 words of dram into
                                r1.
292                                ; (it's important that it's dram due to DMA
                                contention)
293     ;we disable DMA during refresh.
294     move    x:M_DCR2,r2
295     bclr    #M_DE,x:M_DCR2      ;disable the current DMA
296 ;     move    x:(r0)+,r1
297     nop
298     nop
299     nop
300     jset    #M_DACT,x:M_DSTR,*      ;wait for DMA transfer to finish
301
302
303     nop
304     nop
305     nop
306     nop
307     nop
308     nop
309
310     ;depend on BG to tell us when to stop reading.
311     ; we are only returned control when the refresh is done.
312
313     ;we first pulse our refresh pin:
314     bset    #M_DO,x:M_TCSR2
315     bclr    #M_DO,x:M_TCSR2
316
317     ;depend on BG to tell us when to stop reading.
318     ; we are only returned control when the refresh is done.
319 ;     move    x:(r0)+,r1
320 ;     move    x:(r0)+,r1
321
322     nop
323     move    r2,x:M_DCR2
324
325     move    ssh,r2
326     move    ssl,r1
327     move    ssh,r0
328
329
330     rti      ;return; interrupts return with RTI
331
332
333
334 ; Fsetup_keypad_irq
335 ; Functional Specification
336 ;
337 ; Description:      Fsetup_keypad_irq sets up the interrupts for
338 ; the keypad.

```

```

339 ;
340 ; Operation:      It writes the interrupt pin setup registers
341 ; (IPRC) to setup IRQA as a level-triggered pin with priority 1
342 ; (low, we don't care about losing keys too much). We implement
343 ; the interrupt handler with a fast handler (2 opcodes), that are
344 ; copied from memory.
345 ;
346 ; Arguments:      None.
347 ;
348 ; Return Values:  None.
349 ;
350 ; Global Variables: None.
351 ;
352 ; Shared Variables: None.
353 ;
354 ; Local Variables: None.
355 ;
356 ; Inputs:         None.
357 ;
358 ; Outputs:        None.
359 ;
360 ; Error Handling:  None.
361 ;
362 ; Algorithm:       None.
363 ;
364 ; Data Structures: None.
365 ;
366 ; Registers changed: None.
367 ;
368 ; Limitations:     None known.
369 ;
370 ; Known Bugs:      None.
371 ;
372 ; Special Notes:
373 ;
374 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
375 ;
376     global Fsetup_keypad_irq
377 Fsetup_keypad_irq:      ;assembly entry point
378     ;standard C prologue
379     move    #0,n6        ;k is the amount of local space
380     move    ssh,x:(r6)+   ;save the return address
381     move    (r6)+n6      ;allocate local stack space of size k
382
383     ;save all registers we use so we can restore later
384     move    x0,ssh        ;save x0
385     PUSH_ACC a
386
387     ;we first install the interrupt handler
388     move    #>$0bf080,x0  ;unconditional branch to our subroutine
389                        ;(long interrupt)
390     move    x0,p:I_IRQA   ;install handler
391     move    #keypad_handler,x0 ;and install handler address
392     move    x0,p:I_IRQA+1
393
394     ;then we setup the IRQ pin/IRQ registers
395     move    x:M_IPRC,a    ;grab into a so we can operate on it

```

```

396         and    #~M_IAL,a      ;clear the mode mask of IRQA
397         or     #(1<<M_IAL2)|(1<<M_IAL0),a
398                                     ;we set it to be edge-triggered (M_IAL2 = 1)
399                                     ; and priority 1 (M_IAL[0-1] = 1)
400         move    a1,x:M_IPRC    ;and replace back
401
402         ;end of body code, now restore all registers
403         POP_ACC a
404         move    ssh,x0          ;restore x0
405
406         ;standard C epilogue
407         move    #0+1,n6
408         move    (r6)-n6        ;deallocate local stack space, set ccr flags
409         tst     a
410         move    x:(r6),ssh     ;get return address
411
412         nop
413         rts      ;return from subroutine
414 ;end Fsetup_keypad_irq
415
416
417 ; Fend_intr_setup
418 ; Functional Specification
419 ;
420 ; Description:      Fend_intr_setup should be called after setting up interrupts
421 ; (it unmask them).
422 ;
423 ; Operation:       It forces the SR interrupt mask pins to be 0, unmasking
424 ; all interrupts.
425 ;
426 ; Arguments:       None.
427 ;
428 ; Return Values:   None.
429 ;
430 ; Global Variables: None.
431 ;
432 ; Shared Variables: None.
433 ;
434 ; Local Variables: None.
435 ;
436 ; Inputs:          None.
437 ;
438 ; Outputs:         None.
439 ;
440 ; Error Handling:   None.
441 ;
442 ; Algorithm:       None.
443 ;
444 ; Data Structures: None.
445 ;
446 ; Registers changed: None.
447 ;
448 ; Limitations:     None known.
449 ;
450 ; Known Bugs:      None.
451 ;
452 ; Special Notes:

```

```
453 ;
454 ; Revision History: 2011/02/16    Raymond Jimenez    first revision
455 ;
456     global  Fend_intr_setup
457 Fend_intr_setup:                ;C entry point
458     ;standard C prologue
459     move    #0,n6                ;k is the amount of local space
460     move    ssh,x:(r6)+          ;save the return address
461     move    (r6)+n6              ;allocate local stack space of size k
462
463     ;save all registers we use so we can restore later (we don't use any)
464
465     FORCE_INTERRUPTS_ON
466
467     ;end of body code, now restore all registers
468
469     ;standard C epilogue
470     move    #0+1,n6
471     move    (r6)-n6              ;deallocate local stack space, set ccr flags
472     tst     a
473     move    x:(r6),ssh           ;get return address
474
475     nop
476     rts                ;return from subroutine
477 ;end Fend_intr_setup
478
479
480     endsec
481
482
483
```

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
```



```

/*****
/*
/*          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)
=
    /*
    /*
    /* 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();
/*

```

```

while(1)
{
    test_dram();
}
*/

/* first initialize everything */
/* initialize 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 */
    {
        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 definitions 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.

```

1  ;*****
   **
2  ;
3  ; EQUATES for 56302/3 I/O registers and ports
4  ;
5  ; Last update: June 11 1995
6  ;
7  ;*****
   **
8      page 132,55,0,0,0
9      opt mex
10
11  ioequ ident 1,0
12
13  ;-----
14  ;
15  ;   EQUATES for I/O Port Programming
16  ;
17  ;-----
18
19  ; Register Addresses
20  M_HDR      EQU $FFFC9      ; Port B (host) GPIO data Register
21  M_HDDR     EQU $FFFC8      ; Port B (host) GPIO direction Register
22  M_PCRC     EQU $FFFBF      ; Port C (ESSI0) Control Register
23  M_PPRC     EQU $FFFBF      ; Port C (ESSI0) Direction Register
24  M_PDRC     EQU $FFFBF      ; Port C (ESSI0) GPIO Data Register
25  M_PCRD     EQU $FFFAF      ; Port D (ESSI1) Control register
26  M_PPRD     EQU $FFFAE      ; Port D (ESSI1) Direction Data Register
27  M_PDRD     EQU $FFFAF      ; Port D (ESSI1) GPIO Data Register
28  M_PCRE     EQU $FFFF9F     ; Port E (SCI) Control register
29  M_PPRE     EQU $FFFF9E     ; Port E (SCI) Direction Register
30  M_PDRE     EQU $FFFF9D     ; Port E (SCI) Data Register
31  M_OGDB     EQU $FFFFFC     ; OnCE GDB Register
32
33
34  ;-----
35  ;
36  ;   EQUATES for Host Interface
37  ;
38  ;-----
39
40  ; Register Addresses
41  M_HCR      EQU $FFFC2      ; Host Control Register
42  M_HSR      EQU $FFFC3      ; Host Status Register
43  M_HPCR     EQU $FFFC4      ; Host Polarity Control Register
44  M_HBAR     EQU $FFFC5      ; Host Base Address Register
45  M_HRX      EQU $FFFC6      ; Host Receive Register
46  M_HTX      EQU $FFFC7      ; Host Transmit Register
47
48  ; HCR bits definition
49  M_HRIE     EQU $0          ; Host Receive interrupts Enable
50  M_HTIE     EQU $1          ; Host Transmit Interrupt Enable
51  M_HCIE     EQU $2          ; Host Command Interrupt Enable
52  M_HF2      EQU $3          ; Host Flag 2
53  M_HF3      EQU $4          ; Host Flag 3
54
55  ; HSR bits definition

```

```

56 M_HRDF      EQU $0           ; Host Receive Data Full
57 M_HTDE      EQU $1           ; Host Receive Data Emptiy
58 M_HCP       EQU $2           ; Host Command Pending
59 M_HF0       EQU $3           ; Host Flag 0
60 M_HF1       EQU $4           ; Host Flag 1
61
62 ; HPCR bits definition
63 M_HGEN      EQU $0           ; Host Port GPIO Enable
64 M_HA8EN     EQU $1           ; Host Address 8 Enable
65 M_HA9EN     EQU $2           ; Host Address 9 Enable
66 M_HCSEN     EQU $3           ; Host Chip Select Enable
67 M_HREN      EQU $4           ; Host Request Enable
68 M_HAEN      EQU $5           ; Host Acknowledge Enable
69 M_HEN       EQU $6           ; Host Enable
70 M_HOD       EQU $8           ; Host Request Open Drain mode
71 M_HDSP      EQU $9           ; Host Data Strobe Polarity
72 M_HASP      EQU $A           ; Host Address Strobe Polarity
73 M_HMUX      EQU $B           ; Host Multiplexed bus select
74 M_HD_HS     EQU $C           ; Host Double/Single Strobe select
75 M_HCSP      EQU $D           ; Host Chip Select Polarity
76 M_HRP       EQU $E           ; Host Request PolarityPolarity
77 M_HAP       EQU $F           ; Host Acknowledge Polarity
78
79
80 ;-----
81 ;
82 ;   EQUATES for Serial Communications Interface (SCI)
83 ;
84 ;-----
85
86 ; Register Addresses
87 M_STXH      EQU $FFFF97      ; SCI Transmit Data Register (high)
88 M_STXM      EQU $FFFF96      ; SCI Transmit Data Register (middle)
89 M_STXL      EQU $FFFF95      ; SCI Transmit Data Register (low)
90 M_SRXH      EQU $FFFF9A      ; SCI Receive Data Register (high)
91 M_SRXM      EQU $FFFF99      ; SCI Receive Data Register (middle)
92 M_SRXL      EQU $FFFF98      ; SCI Receive Data Register (low)
93 M_STXA      EQU $FFFF94      ; SCI Transmit Address Register
94 M_SCR       EQU $FFFF9C      ; SCI Control Register
95 M_SSR       EQU $FFFF93      ; SCI Status Register
96 M_SCCR      EQU $FFFF9B      ; SCI Clock Control Register
97
98 ; SCI Control Register Bit Flags
99 M_WDS       EQU $7           ; Word Select Mask (WDS0-WDS3)
100 M_WDS0      EQU 0            ; Word Select 0
101 M_WDS1      EQU 1            ; Word Select 1
102 M_WDS2      EQU 2            ; Word Select 2
103 M_SSFTD     EQU 3            ; SCI Shift Direction
104 M_SBK       EQU 4            ; Send Break
105 M_WAKE      EQU 5            ; Wakeup Mode Select
106 M_RWU       EQU 6            ; Receiver Wakeup Enable
107 M_WOMS      EQU 7            ; Wired-OR Mode Select
108 M_SCRE      EQU 8            ; SCI Receiver Enable
109 M_SCTE      EQU 9            ; SCI Transmitter Enable
110 M_ILIE      EQU 10           ; Idle Line Interrupt Enable
111 M_SCRIE     EQU 11           ; SCI Receive Interrupt Enable
112 M_SCTIE     EQU 12           ; SCI Transmit Interrupt Enable

```

```

113 M_TMIE      EQU 13          ; Timer Interrupt Enable
114 M_TIR       EQU 14          ; Timer Interrupt Rate
115 M_SCKP      EQU 15          ; SCI Clock Polarity
116 M_REIE      EQU 16          ; SCI Error Interrupt Enable (REIE)
117
118 ; SCI Status Register Bit Flags
119 M_TRNE      EQU 0           ; Transmitter Empty
120 M_TDRE      EQU 1           ; Transmit Data Register Empty
121 M_RDRF      EQU 2           ; Receive Data Register Full
122 M_IDLE      EQU 3           ; Idle Line Flag
123 M_OR        EQU 4           ; Overrun Error Flag
124 M_PE        EQU 5           ; Parity Error
125 M_FE        EQU 6           ; Framing Error Flag
126 M_R8        EQU 7           ; Received Bit 8 (R8) Address
127
128 ; SCI Clock Control Register
129 M_CD        EQU $FFF        ; Clock Divider Mask (CD0-CD11)
130 M_COD       EQU 12          ; Clock Out Divider
131 M_SCP       EQU 13          ; Clock Prescaler
132 M_RCM       EQU 14          ; Receive Clock Mode Source Bit
133 M_TCM       EQU 15          ; Transmit Clock Source Bit
134
135
136 ;-----
137 ;
138 ;   EQUATES for Enhanced Synchronous Serial Interface (ESSI)
139 ;
140 ;-----
141 ;
142 ; Register Addresses Of ESSIO
143 M_TX00      EQU $FFFFBC     ; ESSIO Transmit Data Register 0
144 M_TX01      EQU $FFFFBB     ; ESSIO Transmit Data Register 1
145 M_TX02      EQU $FFFFBA     ; ESSIO Transmit Data Register 2
146 M_TSR0      EQU $FFFFB9     ; ESSIO Time Slot Register
147 M_RX0       EQU $FFFFB8     ; ESSIO Receive Data Register
148 M_SSISR0    EQU $FFFFB7     ; ESSIO Status Register
149 M_CRB0      EQU $FFFFB6     ; ESSIO Control Register B
150 M_CRA0      EQU $FFFFB5     ; ESSIO Control Register A
151 M_TSMA0     EQU $FFFFB4     ; ESSIO Transmit Slot Mask Register A
152 M_TSMB0     EQU $FFFFB3     ; ESSIO Transmit Slot Mask Register B
153 M_RSMA0     EQU $FFFFB2     ; ESSIO Receive Slot Mask Register A
154 M_RSMB0     EQU $FFFFB1     ; ESSIO Receive Slot Mask Register B
155
156 ; Register Addresses Of ESSII
157 M_TX10      EQU $FFFFAC     ; ESSII Transmit Data Register 0
158 M_TX11      EQU $FFFFAB     ; ESSII Transmit Data Register 1
159 M_TX12      EQU $FFFFAA     ; ESSII Transmit Data Register 2
160 M_TSR1      EQU $FFFFA9     ; ESSII Time Slot Register
161 M_RX1       EQU $FFFFA8     ; ESSII Receive Data Register
162 M_SSISR1    EQU $FFFFA7     ; ESSII Status Register
163 M_CRB1      EQU $FFFFA6     ; ESSII Control Register B
164 M_CRA1      EQU $FFFFA5     ; ESSII Control Register A
165 M_TSMA1     EQU $FFFFA4     ; ESSII Transmit Slot Mask Register A
166 M_TSMB1     EQU $FFFFA3     ; ESSII Transmit Slot Mask Register B
167 M_RSMA1     EQU $FFFFA2     ; ESSII Receive Slot Mask Register A
168 M_RSMB1     EQU $FFFFA1     ; ESSII Receive Slot Mask Register B
169

```

```

170 ; ESSI Control Register A Bit Flags
171 M_PM      EQU $FF      ; Prescale Modulus Select Mask (PM0-PM7)
172 M_PSR     EQU 11       ; Prescaler Range
173 M_DC      EQU $1F000    ; Frame Rate Divider Control Mask (DC0-DC7)
174 M_ALC     EQU 18       ; Alignment Control (ALC)
175 M_WL      EQU $380000   ; Word Length Control Mask (WL0-WL7)
176 M_SSC1    EQU 22       ; Select SC1 as TR #0 drive enable (SSC1)
177
178 ; ESSI Control Register B Bit Flags
179 M_OF      EQU $3        ; Serial Output Flag Mask
180 M_OF0     EQU 0         ; Serial Output Flag 0
181 M_OF1     EQU 1         ; Serial Output Flag 1
182 M_SCD     EQU $1C       ; Serial Control Direction Mask
183 M_SCD0    EQU 2         ; Serial Control 0 Direction
184 M_SCD1    EQU 3         ; Serial Control 1 Direction
185 M_SCD2    EQU 4         ; Serial Control 2 Direction
186 M_SCKD    EQU 5         ; Clock Source Direction
187 M_SHFD    EQU 6         ; Shift Direction
188 M_FSL     EQU $180      ; Frame Sync Length Mask (FSL0-FSL1)
189 M_FSL0    EQU 7         ; Frame Sync Length 0
190 M_FSL1    EQU 8         ; Frame Sync Length 1
191 M_FSR     EQU 9         ; Frame Sync Relative Timing
192 M_FSP     EQU 10        ; Frame Sync Polarity
193 M_CKP     EQU 11        ; Clock Polarity
194 M_SYN     EQU 12        ; Sync/Async Control
195 M_MOD     EQU 13        ; ESSI Mode Select
196 M_SSTE    EQU $1C000    ; ESSI Transmit enable Mask
197 M_SSTE2    EQU 14       ; ESSI Transmit #2 Enable
198 M_SSTE1    EQU 15       ; ESSI Transmit #1 Enable
199 M_SSTE0    EQU 16       ; ESSI Transmit #0 Enable
200 M_SSRE     EQU 17       ; ESSI Receive Enable
201 M_SSTIE    EQU 18       ; ESSI Transmit Interrupt Enable
202 M_SSRIE    EQU 19       ; ESSI Receive Interrupt Enable
203 M_STLIE    EQU 20       ; ESSI Transmit Last Slot Interrupt Enable
204 M_SRLIE    EQU 21       ; ESSI Receive Last Slot Interrupt Enable
205 M_STEIE    EQU 22       ; ESSI Transmit Error Interrupt Enable
206 M_SREIE    EQU 23       ; ESSI Receive Error Interrupt Enable
207
208 ; ESSI Status Register Bit Flags
209 M_IF      EQU $3        ; Serial Input Flag Mask
210 M_IF0     EQU 0         ; Serial Input Flag 0
211 M_IF1     EQU 1         ; Serial Input Flag 1
212 M_TFS     EQU 2         ; Transmit Frame Sync Flag
213 M_RFS     EQU 3         ; Receive Frame Sync Flag
214 M_TUE     EQU 4         ; Transmitter Underrun Error FLAG
215 M_ROE     EQU 5         ; Receiver Overrun Error Flag
216 M_TDE     EQU 6         ; Transmit Data Register Empty
217 M_RDF     EQU 7         ; Receive Data Register Full
218
219 ; ESSI Transmit Slot Mask Register A
220 M_SSTSA    EQU $FFFF    ; ESSI Transmit Slot Bits Mask A (TS0-TS15)
221
222 ; ESSI Transmit Slot Mask Register B
223 M_SSTSB    EQU $FFFF    ; ESSI Transmit Slot Bits Mask B (TS16-TS31)
224
225 ; ESSI Receive Slot Mask Register A
226 M_SSRSA    EQU $FFFF    ; ESSI Receive Slot Bits Mask A (RS0-RS15)

```

```

227
228 ; ESSI Receive Slot Mask Register B
229 M_SSRB EQU $FFFF ; ESSI Receive Slot Bits Mask B (RS16-RS31)
230
231
232 ;-----
233 ;
234 ; EQUATES for Exception Processing
235 ;
236 ;-----
237
238 ; Register Addresses
239 M_IPRC EQU $FFFFFF ; Interrupt Priority Register Core
240 M_IPRP EQU $FFFFFE ; Interrupt Priority Register Peripheral
241
242 ; Interrupt Priority Register Core (IPRC)
243 M_IAL EQU $7 ; IRQA Mode Mask
244 M_IAL0 EQU 0 ; IRQA Mode Interrupt Priority Level (low)
245 M_IAL1 EQU 1 ; IRQA Mode Interrupt Priority Level (high)
246 M_IAL2 EQU 2 ; IRQA Mode Trigger Mode
247 M_IBL EQU $38 ; IRQB Mode Mask
248 M_IBL0 EQU 3 ; IRQB Mode Interrupt Priority Level (low)
249 M_IBL1 EQU 4 ; IRQB Mode Interrupt Priority Level (high)
250 M_IBL2 EQU 5 ; IRQB Mode Trigger Mode
251 M_ICL EQU $1C0 ; IRQC Mode Mask
252 M_ICL0 EQU 6 ; IRQC Mode Interrupt Priority Level (low)
253 M_ICL1 EQU 7 ; IRQC Mode Interrupt Priority Level (high)
254 M_ICL2 EQU 8 ; IRQC Mode Trigger Mode
255 M_IDL EQU $E00 ; IRQD Mode Mask
256 M_IDL0 EQU 9 ; IRQD Mode Interrupt Priority Level (low)
257 M_IDL1 EQU 10 ; IRQD Mode Interrupt Priority Level (high)
258 M_IDL2 EQU 11 ; IRQD Mode Trigger Mode
259 M_D0L EQU $3000 ; DMA0 Interrupt priority Level Mask
260 M_D0L0 EQU 12 ; DMA0 Interrupt Priority Level (low)
261 M_D0L1 EQU 13 ; DMA0 Interrupt Priority Level (high)
262 M_D1L EQU $C000 ; DMA1 Interrupt Priority Level Mask
263 M_D1L0 EQU 14 ; DMA1 Interrupt Priority Level (low)
264 M_D1L1 EQU 15 ; DMA1 Interrupt Priority Level (high)
265 M_D2L EQU $30000 ; DMA2 Interrupt priority Level Mask
266 M_D2L0 EQU 16 ; DMA2 Interrupt Priority Level (low)
267 M_D2L1 EQU 17 ; DMA2 Interrupt Priority Level (high)
268 M_D3L EQU $C0000 ; DMA3 Interrupt Priority Level Mask
269 M_D3L0 EQU 18 ; DMA3 Interrupt Priority Level (low)
270 M_D3L1 EQU 19 ; DMA3 Interrupt Priority Level (high)
271 M_D4L EQU $300000 ; DMA4 Interrupt priority Level Mask
272 M_D4L0 EQU 20 ; DMA4 Interrupt Priority Level (low)
273 M_D4L1 EQU 21 ; DMA4 Interrupt Priority Level (high)
274 M_D5L EQU $C00000 ; DMA5 Interrupt priority Level Mask
275 M_D5L0 EQU 22 ; DMA5 Interrupt Priority Level (low)
276 M_D5L1 EQU 23 ; DMA5 Interrupt Priority Level (high)
277
278 ; Interrupt Priority Register Peripheral (IPRP)
279 M_HPL EQU $3 ; Host Interrupt Priority Level Mask
280 M_HPL0 EQU 0 ; Host Interrupt Priority Level (low)
281 M_HPL1 EQU 1 ; Host Interrupt Priority Level (high)
282 M_S0L EQU $C ; ESSI0 Interrupt Priority Level Mask
283 M_S0L0 EQU 2 ; ESSI0 Interrupt Priority Level (low)

```

```

284 M_S0L1      EQU 3          ; ESSIO Interrupt Priority Level (high)
285 M_S1L       EQU $30        ; ESSIO Interrupt Priority Level Mask
286 M_S1L0      EQU 4          ; ESSIO Interrupt Priority Level (low)
287 M_S1L1      EQU 5          ; ESSIO Interrupt Priority Level (high)
288 M_SCL       EQU $C0        ; SCI Interrupt Priority Level Mask
289 M_SCL0      EQU 6          ; SCI Interrupt Priority Level (low)
290 M_SCL1      EQU 7          ; SCI Interrupt Priority Level (high)
291 M_T0L       EQU $300        ; TIMER Interrupt Priority Level Mask
292 M_T0L0      EQU 8          ; TIMER Interrupt Priority Level (low)
293 M_T0L1      EQU 9          ; TIMER Interrupt Priority Level (high)
294
295
296 ;-----
297 ;
298 ;   EQUATES for TIMER
299 ;
300 ;-----
301
302 ; Register Addresses Of TIMER0
303 M_TCSR0      EQU $FFFF8F    ; TIMER0 Control/Status Register
304 M_TLR0       EQU $FFFF8E    ; TIMER0 Load Reg
305 M_TCPR0      EQU $FFFF8D    ; TIMER0 Compare Register
306 M_TCR0       EQU $FFFF8C    ; TIMER0 Count Register
307
308 ; Register Addresses Of TIMER1
309 M_TCSR1      EQU $FFFF8B    ; TIMER1 Control/Status Register
310 M_TLR1       EQU $FFFF8A    ; TIMER1 Load Reg
311 M_TCPR1      EQU $FFFF89    ; TIMER1 Compare Register
312 M_TCR1       EQU $FFFF88    ; TIMER1 Count Register
313
314 ; Register Addresses Of TIMER2
315 M_TCSR2      EQU $FFFF87    ; TIMER2 Control/Status Register
316 M_TLR2       EQU $FFFF86    ; TIMER2 Load Reg
317 M_TCPR2      EQU $FFFF85    ; TIMER2 Compare Register
318 M_TCR2       EQU $FFFF84    ; TIMER2 Count Register
319 M_TPLR       EQU $FFFF83    ; TIMER Prescaler Load Register
320 M_TPCR       EQU $FFFF82    ; TIMER Prescaler Count Register
321
322 ; Timer Control/Status Register Bit Flags
323 M_TE         EQU 0          ; Timer Enable
324 M_TOIE       EQU 1          ; Timer Overflow Interrupt Enable
325 M_TCIE       EQU 2          ; Timer Compare Interrupt Enable
326 M_TC         EQU $F0        ; Timer Control Mask (TC0-TC3)
327 M_INV        EQU 8          ; Inverter Bit
328 M_TRM        EQU 9          ; Timer Restart Mode
329 M_DIR        EQU 11         ; Direction Bit
330 M_DI         EQU 12         ; Data Input
331 M_DO         EQU 13         ; Data Output
332 M_PCE        EQU 15         ; Prescaled Clock Enable
333 M_TOF        EQU 20         ; Timer Overflow Flag
334 M_TCF        EQU 21         ; Timer Compare Flag
335
336 ; Timer Prescaler Register Bit Flags
337 M_PS         EQU $600000    ; Prescaler Source Mask
338 M_PS0        EQU 21
339 M_PS1        EQU 22
340

```



```

341 ; Timer Control Bits
342 M_TC0      EQU 4           ; Timer Control 0
343 M_TC1      EQU 5           ; Timer Control 1
344 M_TC2      EQU 6           ; Timer Control 2
345 M_TC3      EQU 7           ; Timer Control 3
346
347
348 ;-----
349 ;
350 ;   EQUATES for Direct Memory Access (DMA)
351 ;
352 ;-----
353
354 ; Register Addresses Of DMA
355 M_DSTR      EQU $FFFFFF4    ; DMA Status Register
356 M_DOR0      EQU $FFFFFF3    ; DMA Offset Register 0
357 M_DOR1      EQU $FFFFFF2    ; DMA Offset Register 1
358 M_DOR2      EQU $FFFFFF1    ; DMA Offset Register 2
359 M_DOR3      EQU $FFFFFF0    ; DMA Offset Register 3
360
361 ; Register Addresses Of DMA0
362 M_DSR0      EQU $FFFFEF      ; DMA0 Source Address Register
363 M_DDR0      EQU $FFFFEE      ; DMA0 Destination Address Register
364 M_DC00      EQU $FFFFED      ; DMA0 Counter
365 M_DCR0      EQU $FFFFEC      ; DMA0 Control Register
366
367 ; Register Addresses Of DMA1
368 M_DSR1      EQU $FFFFEB      ; DMA1 Source Address Register
369 M_DDR1      EQU $FFFFEA      ; DMA1 Destination Address Register
370 M_DC01      EQU $FFFFE9      ; DMA1 Counter
371 M_DCR1      EQU $FFFFE8      ; DMA1 Control Register
372
373 ; Register Addresses Of DMA2
374 M_DSR2      EQU $FFFFE7      ; DMA2 Source Address Register
375 M_DDR2      EQU $FFFFE6      ; DMA2 Destination Address Register
376 M_DC02      EQU $FFFFE5      ; DMA2 Counter
377 M_DCR2      EQU $FFFFE4      ; DMA2 Control Register
378
379 ; Register Addresses Of DMA3
380 M_DSR3      EQU $FFFFE3      ; DMA3 Source Address Register
381 M_DDR3      EQU $FFFFE2      ; DMA3 Destination Address Register
382 M_DC03      EQU $FFFFE1      ; DMA3 Counter
383 M_DCR3      EQU $FFFFE0      ; DMA3 Control Register
384
385 ; Register Addresses Of DMA4
386 M_DSR4      EQU $FFFFDF      ; DMA4 Source Address Register
387 M_DDR4      EQU $FFFFDE      ; DMA4 Destination Address Register
388 M_DC04      EQU $FFFFDD      ; DMA4 Counter
389 M_DCR4      EQU $FFFFDC      ; DMA4 Control Register
390
391 ; Register Addresses Of DMA5
392 M_DSR5      EQU $FFFFDB      ; DMA5 Source Address Register
393 M_DDR5      EQU $FFFFDA      ; DMA5 Destination Address Register
394 M_DC05      EQU $FFFFD9      ; DMA5 Counter
395 M_DCR5      EQU $FFFFD8      ; DMA5 Control Register
396
397 ; DMA Control Register

```

```

398 M_DSS      EQU $3          ; DMA Source Space Mask (DSS0-Dss1)
399 M_DSS0     EQU 0           ; DMA Source Memory space 0
400 M_DSS1     EQU 1           ; DMA Source Memory space 1
401 M_DDS      EQU $C          ; DMA Destination Space Mask (DDS-DDS1)
402 M_DDS0     EQU 2           ; DMA Destination Memory Space 0
403 M_DDS1     EQU 3           ; DMA Destination Memory Space 1
404 M_DAM      EQU $3f0        ; DMA Address Mode Mask (DAM5-DAM0)
405 M_DAM0     EQU 4           ; DMA Address Mode 0
406 M_DAM1     EQU 5           ; DMA Address Mode 1
407 M_DAM2     EQU 6           ; DMA Address Mode 2
408 M_DAM3     EQU 7           ; DMA Address Mode 3
409 M_DAM4     EQU 8           ; DMA Address Mode 4
410 M_DAM5     EQU 9           ; DMA Address Mode 5
411 M_D3D      EQU 10          ; DMA Three Dimensional Mode
412 M_DRS      EQU $F800       ; DMA Request Source Mask (DRS0-DRS4)
413 M_DCON     EQU 16          ; DMA Continuous Mode
414 M_DPR      EQU $60000      ; DMA Channel Priority
415 M_DPR0     EQU 17          ; DMA Channel Priority Level (low)
416 M_DPR1     EQU 18          ; DMA Channel Priority Level (high)
417 M_DTM      EQU $380000     ; DMA Transfer Mode Mask (DTM2-DTM0)
418 M_DTM0     EQU 19          ; DMA Transfer Mode 0
419 M_DTM1     EQU 20          ; DMA Transfer Mode 1
420 M_DTM2     EQU 21          ; DMA Transfer Mode 2
421 M_DIE      EQU 22          ; DMA Interrupt Enable bit
422 M_DE       EQU 23          ; DMA Channel Enable bit
423
424 ; DMA Status Register
425 M_DTD      EQU $3F         ; Channel Transfer Done Status MASK (DTD0-DTD5)
426 M_DTD0     EQU 0           ; DMA Channel Transfer Done Status 0
427 M_DTD1     EQU 1           ; DMA Channel Transfer Done Status 1
428 M_DTD2     EQU 2           ; DMA Channel Transfer Done Status 2
429 M_DTD3     EQU 3           ; DMA Channel Transfer Done Status 3
430 M_DTD4     EQU 4           ; DMA Channel Transfer Done Status 4
431 M_DTD5     EQU 5           ; DMA Channel Transfer Done Status 5
432 M_DACT     EQU 8           ; DMA Active State
433 M_DCH      EQU $E00        ; DMA Active Channel Mask (DCH0-DCH2)
434 M_DCH0     EQU 9           ; DMA Active Channel 0
435 M_DCH1     EQU 10          ; DMA Active Channel 1
436 M_DCH2     EQU 11          ; DMA Active Channel 2
437
438
439 ;-----
440 ;
441 ;   EQUATES for Phase Locked Loop (PLL)
442 ;
443 ;-----
444
445 ; Register Addresses Of PLL
446 M_PCTL      EQU $FFFFFF     ; PLL Control Register
447
448 ; PLL Control Register
449 M_MF        EQU $FFF        ; Multiplication Factor Bits Mask (MF0-MF11)
450 M_DF        EQU $7000       ; Division Factor Bits Mask (DF0-DF2)
451 M_XTLR      EQU 15          ; XTAL Range select bit
452 M_XTLD      EQU 16          ; XTAL Disable Bit
453 M_PSTP      EQU 17          ; STOP Processing State Bit
454 M_PEN       EQU 18          ; PLL Enable Bit

```

```

455 M_PCOD      EQU 19          ; PLL Clock Output Disable Bit
456 M_PD        EQU $F00000    ; PreDivider Factor Bits Mask (PD0-PD3)
457
458
459 ;-----
460 ;
461 ;   EQUATES for BIU
462 ;
463 ;-----
464
465 ; Register Addresses Of BIU
466 M_BCR        EQU $FFFFFB    ; Bus Control Register
467 M_DCR        EQU $FFFFFA    ; DRAM Control Register
468 M_AAR0       EQU $FFFFF9    ; Address Attribute Register 0
469 M_AAR1       EQU $FFFFF8    ; Address Attribute Register 1
470 M_AAR2       EQU $FFFFF7    ; Address Attribute Register 2
471 M_AAR3       EQU $FFFFF6    ; Address Attribute Register 3
472 M_IDR        EQU $FFFFF5    ; ID Register
473
474 ; Bus Control Register
475 M_BA0W       EQU $1F        ; Area 0 Wait Control Mask (BA0W0-BA0W4)
476 M_BA1W       EQU $3E0      ; Area 1 Wait Control Mask (BA1W0-BA1W4)
477 M_BA2W       EQU $1C00     ; Area 2 Wait Control Mask (BA2W0-BA2W4)
478 M_BA3W       EQU $E000     ; Area 3 Wait Control Mask (BA3W0-BA3W4)
479 M_BDFW       EQU $1F0000    ; Default Area Wait Control Mask (BDFW0-BDFW4)
480 M_BBS        EQU 21        ; Bus State
481 M_BLH        EQU 22        ; Bus Lock Hold
482 M_BRH        EQU 23        ; Bus Request Hold
483
484 ; DRAM Control Register
485 M_BCW        EQU $3         ; In Page Wait States Bits Mask (BCW0-BCW1)
486 M_BRW        EQU $C         ; Out Of Page Wait States Bits Mask (BRW0-BRW1)
487 M_BPS        EQU $300      ; DRAM Page Size Bits Mask (BPS0-BPS1)
488 M_BPLE       EQU 11        ; Page Logic Enable
489 M_BME        EQU 12        ; Mastership Enable
490 M_BRE        EQU 13        ; Refresh Enable
491 M_BSTR       EQU 14        ; Software Triggered Refresh
492 M_BRF        EQU $7F8000    ; Refresh Rate Bits Mask (BRF0-BRF7)
493 M_BRP        EQU 23        ; Refresh prescaler
494
495 ; Address Attribute Registers
496 M_BAT        EQU $3         ; External Access Type and Pin Definition Bits
Mask(BAT0-BAT1)
497 M_BAAP       EQU 2         ; Address Attribute Pin Polarity
498 M_BPEN       EQU 3         ; Program Space Enable
499 M_BXEN       EQU 4         ; X Data Space Enable
500 M_BYEN       EQU 5         ; Y Data Space Enable
501 M_BAM        EQU 6         ; Address Muxing
502 M_BPAC       EQU 7         ; Packing Enable
503 M_BNC        EQU $F00      ; Number of Address Bits to Compare
Mask(BNC0-BNC3)
504 M_BAC        EQU $FFF000    ; Address to Compare Bits Mask (BAC0-BAC11)
505
506 ; control and status bits in SR
507 M_CP         EQU $c00000    ; mask for CORE-DMA priority bits in SR
508 M_CA         EQU 0         ; Carry
509 M_V          EQU 1         ; Overflow

```

```

510 M_Z      EQU 2      ; Zero
511 M_N      EQU 3      ; Negative
512 M_U      EQU 4      ; Unnormalized
513 M_E      EQU 5      ; Extension
514 M_L      EQU 6      ; Limit
515 M_S      EQU 7      ; Scaling Bit
516 M_I0     EQU 8      ; Interupt Mask Bit 0
517 M_I1     EQU 9      ; Interupt Mask Bit 1
518 M_S0     EQU 10     ; Scaling Mode Bit 0
519 M_S1     EQU 11     ; Scaling Mode Bit 1
520 M_SC      EQU 13     ; Sixteen_Bit Compatibility
521 M_DM      EQU 14     ; Double Precision Multiply
522 M_LF      EQU 15     ; DO-Loop Flag
523 M_FV      EQU 16     ; DO-Forever Flag
524 M_SA      EQU 17     ; Sixteen-Bit Arithmetic
525 M_CE      EQU 19     ; Instruction Cache Enable
526 M_SM      EQU 20     ; Arithmetic Saturation
527 M_RM      EQU 21     ; Rounding Mode
528 M_CP0     EQU 22     ; bit 0 of priority bits in SR
529 M_CP1     EQU 23     ; bit 1 of priority bits in SR
530
531 ; control and status bits in OMR
532 M_CDP      EQU $300   ; mask for CORE-DMA priority bits in OMR
533 M_MA       EQU 0      ; Operating Mode A
534 M_MB       EQU 1      ; Operating Mode B
535 M_MC       EQU 2      ; Operating Mode C
536 M_MD       EQU 3      ; Operating Mode D
537 M_EBD      EQU 4      ; External Bus Disable bit in OMR
538 M_SD       EQU 6      ; Stop Delay
539 M_MS       EQU 7      ; Memory Switch bit in OMR
540 M_CDP0     EQU 8      ; bit 0 of priority bits in OMR
541 M_CDP1     EQU 9      ; bit 1 of priority bits in OMR
542 M_BEN      EQU 10     ; Burst Enable
543 M_TAS      EQU 11     ; TA Synchronize Select
544 M_BRT      EQU 12     ; Bus Release Timing
545 M_ATE      EQU 15     ; Address Tracing Enable bit in OMR.
546 M_XYS      EQU 16     ; Stack Extension space select bit in OMR.
547 M_EUN      EQU 17     ; Extended stack Underflow flag in OMR.
548 M_EOV      EQU 18     ; Extended stack Overflow flag in OMR.
549 M_WRP      EQU 19     ; Extended WRaP flag in OMR.
550 M_SEN      EQU 20     ; Stack Extension Enable bit in OMR.
551
552
553 ;*****
554 ;
555 ;   EQUATES for 56302/3 interrupts
556 ;
557 ; Last update: June 11 1995
558 ;
559 ;*****
560
560     page 132,55,0,0,0
561     opt mex
562     intequ ident 1,0
563     if @DEF(I_VEC)
564
;leave user definition as is.

```

```

565     else
566 I_VEC      EQU $0
567     endif
568
569 ;-----
570 ; Non-Maskable interrupts
571 ;-----
572 I_RESET    EQU I_VEC+$00      ; Hardware RESET
573 I_STACK    EQU I_VEC+$02      ; Stack Error
574 I_ILL      EQU I_VEC+$04      ; Illegal Instruction
575 I_DBG      EQU I_VEC+$06      ; Debug Request
576 I_TRAP     EQU I_VEC+$08      ; Trap
577 I_NMI      EQU I_VEC+$0A      ; Non Maskable Interrupt
578
579 ;-----
580 ; Interrupt Request Pins
581 ;-----
582 I_IRQA     EQU I_VEC+$10      ; IRQA
583 I_IRQB     EQU I_VEC+$12      ; IRQB
584 I_IRQC     EQU I_VEC+$14      ; IRQC
585 I_IRQD     EQU I_VEC+$16      ; IRQD
586
587 ;-----
588 ; DMA Interrupts
589 ;-----
590 I_DMA0     EQU I_VEC+$18      ; DMA Channel 0
591 I_DMA1     EQU I_VEC+$1A      ; DMA Channel 1
592 I_DMA2     EQU I_VEC+$1C      ; DMA Channel 2
593 I_DMA3     EQU I_VEC+$1E      ; DMA Channel 3
594 I_DMA4     EQU I_VEC+$20      ; DMA Channel 4
595 I_DMA5     EQU I_VEC+$22      ; DMA Channel 5
596
597 ;-----
598 ; Timer Interrupts
599 ;-----
600 I_TIM0C    EQU I_VEC+$24      ; TIMER 0 compare
601 I_TIM0OF   EQU I_VEC+$26      ; TIMER 0 overflow
602 I_TIM1C    EQU I_VEC+$28      ; TIMER 1 compare
603 I_TIM1OF   EQU I_VEC+$2A      ; TIMER 1 overflow
604 I_TIM2C    EQU I_VEC+$2C      ; TIMER 2 compare
605 I_TIM2OF   EQU I_VEC+$2E      ; TIMER 2 overflow
606
607 ;-----
608 ; ESSI Interrupts
609 ;-----
610 I_SI0RD    EQU I_VEC+$30      ; ESSI0 Receive Data
611 I_SI0RDE   EQU I_VEC+$32      ; ESSI0 Receive Data With Exception Status
612 I_SI0RLS   EQU I_VEC+$34      ; ESSI0 Receive last slot
613 I_SI0TD    EQU I_VEC+$36      ; ESSI0 Transmit data
614 I_SI0TDE   EQU I_VEC+$38      ; ESSI0 Transmit Data With Exception Status
615 I_SI0TLS   EQU I_VEC+$3A      ; ESSI0 Transmit last slot
616 I_SI1RD    EQU I_VEC+$40      ; ESSI1 Receive Data
617 I_SI1RDE   EQU I_VEC+$42      ; ESSI1 Receive Data With Exception Status
618 I_SI1RLS   EQU I_VEC+$44      ; ESSI1 Receive last slot
619 I_SI1TD    EQU I_VEC+$46      ; ESSI1 Transmit data
620 I_SI1TDE   EQU I_VEC+$48      ; ESSI1 Transmit Data With Exception Status
621 I_SI1TLS   EQU I_VEC+$4A      ; ESSI1 Transmit last slot

```

```
622
623 ;-----
624 ; SCI Interrupts
625 ;-----
626 I_SCIRD      EQU I_VEC+$50      ; SCI Receive Data
627 I_SCIRDE     EQU I_VEC+$52      ; SCI Receive Data With Exception Status
628 I_SCITD      EQU I_VEC+$54      ; SCI Transmit Data
629 I_SCIIL      EQU I_VEC+$56      ; SCI Idle Line
630 I_SCITM      EQU I_VEC+$58      ; SCI Timer
631
632 ;-----
633 ; HOST Interrupts
634 ;-----
635 I_HRDF       EQU I_VEC+$60      ; Host Receive Data Full
636 I_HTDE       EQU I_VEC+$62      ; Host Transmit Data Empty
637 I_HC         EQU I_VEC+$64      ; Default Host Command
638
639 ;-----
640 ; INTERRUPT ENDING ADDRESS
641 ;-----
642 I_INTEND     EQU I_VEC+$FF      ; last address of interrupt vector space
643
644
```

```
1 FALSE EQU 0
2 TRUE EQU 1
3
```

```

1  ; we place some convenient macros here to make our life easier:
2
3  PUSH_ACC      MACRO    ACC
4
5      ;we're saving an accumulator to the stack.
6      move      ACC\0,ssh      ;push acc0/acc1 onto stack in one location
7      move      ACC\1,ssl      ;pushing acc0 already preincremented
8      move      ACC\2,ssh      ; and then use another level for acc2
9
10     ENDM
11
12  POP_ACC       MACRO    ACC
13
14     ;we're retrieving an accumulator from the stack.
15     move      ssh,ACC\2
16     move      ssl,ACC\1      ;move acc1->acc1
17     move      ssh,ACC\0      ;finish off with ACC1
18
19     ENDM
20
21  START_CRITICAL MACRO
22     ;grab the interrupt bits and toss it on the stack
23     movec     sr,ssh          ;toss it on the stack
24     ori      #$03,mr          ;set the interrupt mask bits
25     nop
26     nop
27     nop
28
29     ;-- START OF CRITICAL CODE --
30     ENDM
31
32  END_CRITICAL   MACRO
33     ; -- END OF CRITICAL CODE --
34     ;then reset the interrupt bits to normal
35     nop
36     nop
37     nop
38
39     movec     ssh,sr          ;back to normal
40     ENDM
41
42  FORCE_INTERRUPTS_ON MACRO
43     ;we force interrupts on
44     andi      #$fc,mr          ;set SR mask bits to 0, unmasking all interrupts
45     ENDM
46
47  FORCE_INTERRUPTS_OFF MACRO
48     ;we force interrupts off
49     ori      #$03,mr          ;set SR mask bits to 3, masking all interrupts
50     ENDM
51

```



```
#####
#
# 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=dsp1nk
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 $(JUKEDIR)/*.c)
JUKEOBS=$(notdir $(JUKEFILES:.c=.cln))

# Linker Command File

LINKCMD=jukebox.cli

# low-level assembly files

# user must define the symbol SYSOBS below
# example definition:
# SYSOBS= keypad.cln display.cln
SYSOBS= 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) $(JUKEOBS)
#      $(LINK) -f$(LINKCMD)
#      $(HEXMAKE) -b jukebox.cld
```

```
jukebox.cld: $(SYSOBJS) $(JUKEOBS)
    $(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 definitions:
#   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 point 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 bandwidth 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/>