

Monophonic Pitch Recognition

A Senior Project

presented to

the Faculty of the Electrical Engineering Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Electrical Engineering

by

Nathan Zorndorf and Kristine Carreon

June, 2013

Table of Contents

List of Figures	3
List of Tables	4
Acknowledgements	5
Abstract.....	6
Introduction	6
Background	7
Pitch Identification.....	8
Requirements.....	11
Functional Requirements:	11
Performance Specifications:	11
Design Approaches	11
Approaches Considered	11
Anti-aliasing filter topology.....	14
Project Design	16
Description of Subsystems and Theory of Operation	18
Physical Construction and Integration	19
Integrated System Tests and Results.....	19
Conclusions	22
Bibliography	24
Appendices.....	25
Specifications.....	25
Schedule	26
Program Listing	27
Memory Mapping	81

List of Figures

FIGURE 1: FUNDAMENTAL FREQUENCY VALUE RANGE IN WESTERN MUSIC USING EQUAL TEMPERAMENT TUNING(FEILDING).	7
FIGURE 2: A FREQUENCY DOMAIN REPRESENTATION OF A GUITAR PLAYING THE NOTE D4. INSTRUMENTS THAT PLAY A PITCH USUALLY HAVE HARMONICS ASSOCIATED WITH THEIR FUNDAMENTALS IN THEIR SPECTRUMS. THE FIRST SPIKE IN MAGNITUDE APPEARS AT 293.7 Hz, WITH HARMONICS APPEARING PAST 3000 Hz (CHAPMAN).	8
FIGURE 3: A TIME DOMAIN REPRESENTATION OF A NOTE PLAYED BY A FLUTE (FITZ).....	8
FIGURE 4: DTFT	10
FIGURE 5: FFT.....	10
FIGURE 6: IFFT.....	10
FIGURE 7: RESULTS FOR THE TIMING TEST FOR THE FFT TO FIRST PEAK SEARCH ALGORITHM.	12
FIGURE 8: FFT OF A RECORDER PLAYING Eb, AND THE RESULTS OF THE PEACK SEARCH.....	13
FIGURE 9: : PROCESSING OF FIVE CONTIGUOUS NOTES.	14
FIGURE 10: ANTI-ALIASING SCHEMATIC WITH A CUTOFF FREQUENCY OF 4 KHZ.....	14
FIGURE 11: ANTI-ALIASING FILTER SIMULATED FREQUENCY RESPONSE.	14
FIGURE 12: MEASURED FREQUENCY RESPONSE OF FILTER.....	15
FIGURE 13: A HIGH LEVEL REPRESENTATIONAL DIAGRAM FOR THE SYSTEM.....	16
FIGURE 14: BLACK BOX DIAGRAM OF THE SYSTEM.....	16
FIGURE 15: COMPLETE SYSTEM BLOCK DIAGRAM.	17
FIGURE 16: PHYSICAL IMPLEMENTATION. TOP LEFT AUDIO JACK IS MICROPHONE INPUT. SIGNAL RUNS THROUGH FILTER AND IS INPUT TO AUDIO JACK ON DSP BOARD. AFTER PROCESSING THIS DATA, MIDI OUTPUT IS SENT THROUGH CONNECTOR PIN TO MIDI INTERFACING HARDWARE.	19
FIGURE 19: EXPECTED RESULTS FROM THE THIRD TEST TRIAL INVOLVING A RECORDING OF A FLUTE PLAYING MIDI NOTES 84, 86, AND 88.	21
FIGURE 20: ACTUAL EXPERIMENTAL RESULTS OF THE THIRD TEST TRIAL. THE LONGER SUSTAINED NOTES ARE CORRECT, BUT THE SMALL MIDI NOTES ARE INCORRECT PITCHES BEING IDENTIFIED.	21
FIGURE 21: WORK SCHEDULE FOR THE PROJECT.	26

List of Tables

TABLE 1: SUMMARY OF TEST RESULTS FOR THE TRIAL INVOLVING 5 CONTIGUOUS FLUTE NOTES.20

TABLE 2: MAXIMUM ERROR FROM THE THREE TEST CASES INVOLVING FIVE CONTIGUOUS FLUTE NOTES.....20

Acknowledgements

Special thanks to our project advisor Dr. Wayne Pilkington, and industry advisors Ton Kalker and Douglas Mandell.

Abstract

The purpose of this project is to create a system that automatically converts monophonic music into its MIDI equivalent. Automatic pitch recognition allows for numerous commercial applications, including automatic transcription and digital storage of live performances. It is also desirable to be able to take an audio signal as an input and create a MIDI equivalent score because the MIDI information can be used to replace the original audio signal sounds with any sound the user would like. For example, if a piano composition is entered into the system, the resulting MIDI out could be used to trigger guitar samples.

The main deliverable for this project is a DSP evaluation board that takes a monophonic analog audio signal (ex. a recorder or someone's voice creating one pitch at a time), analyzes the signal for its fundamental frequency, and outputs MIDI data that represents the pitch and timing information contained in the audio signal - all in real time.

Introduction

Our main motivation for this project was our love of both technology and music. As electrical engineering majors and music minors, we both had technical abilities with a musical inclination. We shared the feeling of inconvenience when transcribing music and wanted to develop something that could keep a record of what notes were being played. We knew that if we were able to recognize pitches and timing information, the information would be most easily transmittable using the MIDI specification, being the de facto standard for communication musical information among electronic instruments.

This technology has several music-related applications and is meant for musicians of all skill levels. It can be used by musicians who want to transcribe music as it is played and keep a digital record of their playing. This is useful for musicians that do not know how to notate music. It could also be useful in recording live jams where much of the music is improvised. One could also easily perform pitch corrections/alterations on MIDI data. MIDI is also useful in that you can apply different timbres to pitches (ex. use your voice to produce a guitar sound). This would allow someone who can only play one instrument, or who can only sing, the ability to produce sounds of all types of instruments. MIDI itself is so widely used: our project can cooperate with any device or software that is MIDI compatible.

Other pitch recognition software does exist, the most popular being Celemony's Melodyne. Alternative software has been known to be mediocre. The downside of Melodyne is the cost. It is a huge software package with many features, most of which would be overwhelming and unnecessary to novices or those not interested in production. Our solution is focused on audio-MIDI conversion. One feature that our product is unique in is its ability to perform audio-MIDI conversion in real-time. Similar existing software intakes an audio recording whereas our project can intake live audio. Our particular project is meant as a proof of concept. Implementing a

marketable solution would involve designing a PC board to interface the DSP chip, necessary peripherals, and the MIDI interfacing circuitry.

Background

Music Theory

Equal tempered music is made up of audible musical pitches whose fundamental frequencies range from 20.60 Hz (the pitch E_0 – meaning the note E located within in the 0th octave) to 4979 Hz (the pitch $D_8^\#$ – meaning the note D sharp in the 8th octave). Equal-temperament is a system of tuning (allotting note names to frequencies) where each pair of adjacent pitches has the same frequency ratio. This causes each octave (where the distance between two pitches an octave apart is defined as twice the frequency of the lower pitch) to be comprised of 12 equally spaced pitches. Figure 1 shows how equal temperament splits up each octave.

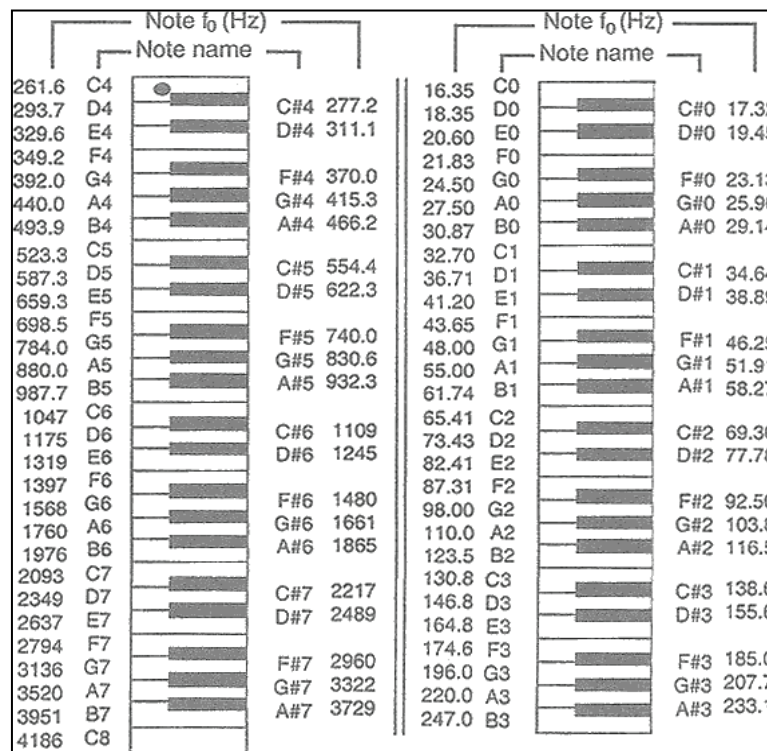


Figure 1: Fundamental Frequency Value Range in Western Music using Equal Temperament Tuning(Feilding).

Representations of Pitch in the Time and Frequency Domain

Musical pitches are made up of one fundamental frequency and harmonics related to that fundamental. Fourier transforms allow us to observe the frequency content contained in an audio signal. The frequency domain representation of a single pitch (D_4) created by a guitar is shown in Figure 2 below.

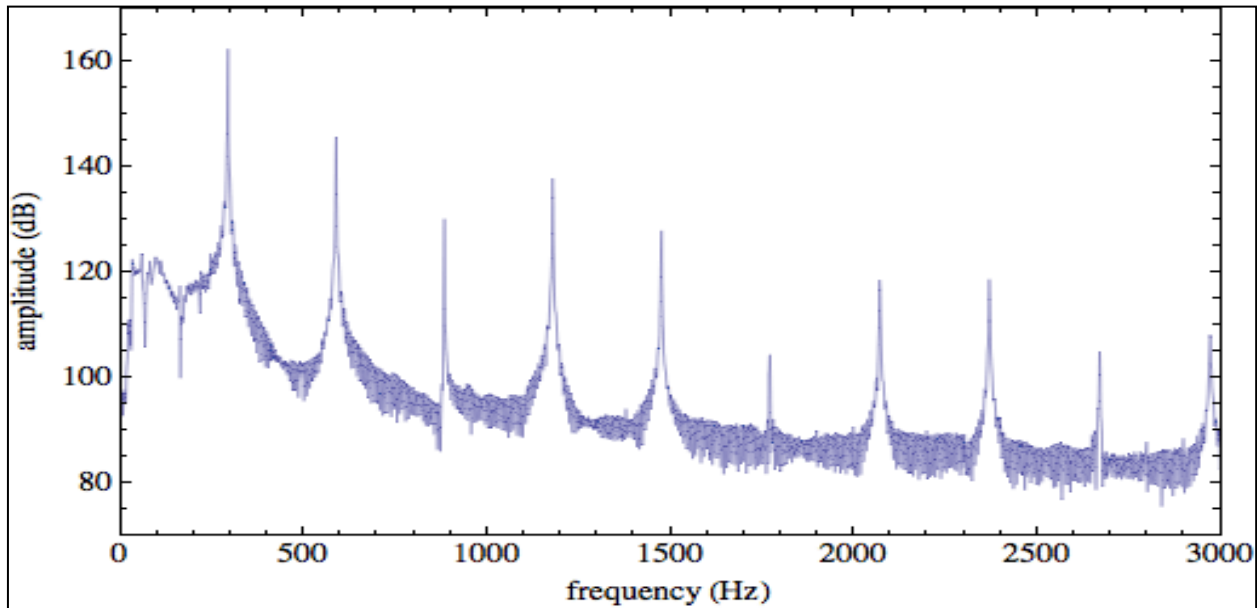


Figure 2: A frequency domain representation of a guitar playing the note D4. Instruments that play a pitch usually have harmonics associated with their fundamentals in their spectrums. The first spike in magnitude appears at 293.7 Hz, with harmonics appearing past 3000 Hz (Chapman).

A time domain representation of a pitch played by a flute is shown in figure 3 below. Notice that the note changes with time in an envelope that can be characterized as having an attack, sustain, and release portion. The significance of these portions in terms of pitch identification will be discussed later.

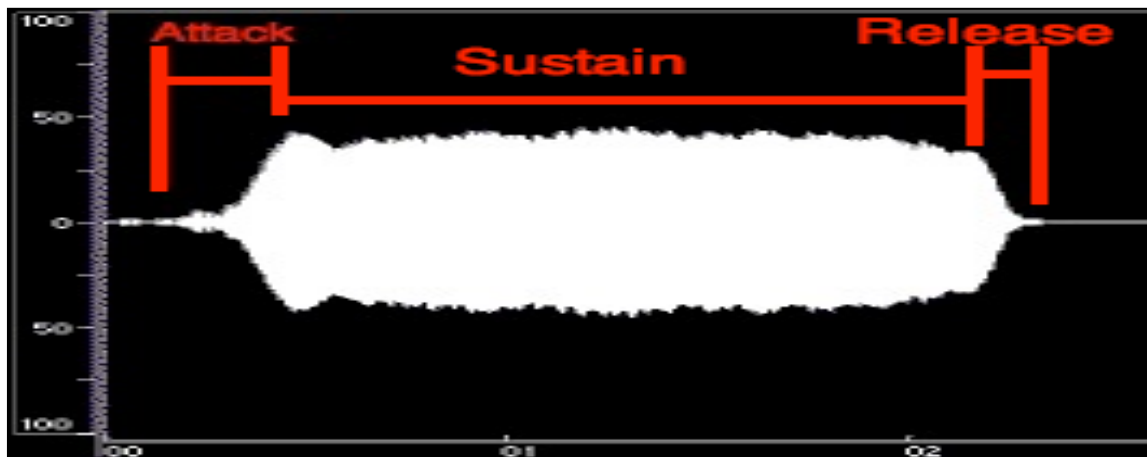


Figure 3: A time domain representation of a note played by a flute (Fitz).

Pitch Identification

Monophonic frequency identification refers to recognizing what pitches are contained in a monophonic audio signal when only one pitch is produced at a time. When several notes are played at once, polyphonic frequency identification identifies each of those notes. A polyphonic signal is made up of several signals (each with their own harmonics) which combine to form a composite signal. The composite spectrum of this composite signal contains overlapping

harmonics, making it difficult to decide which fundamental frequency a harmonic applies to. These polyphonic signals could come from several monophonic instruments playing at once, or a polyphonic instrument generating several pitches at once. The goal of pitch identification is to identify all of the fundamental frequencies that are contained within an audio signal. Polyphonic note identification has to this date, not seen a complete, infallible solution, although software named Melodyne does come close. The fundamental goal behind *this* project is monophonic pitch identification for audio produced by a recorder.

Current Monophonic Note Recognition Algorithms

- **Time Domain Autocorrelation** – Time domain methods of pitch identification focus on discovering information relating to the periodicity of the input signal. This algorithm is generally phase insensitive and easily ported to hardware. An estimate of the autocorrelation of an N-Length digital signal $x(n)$ is given by

$$r_{xx}(n) = \frac{1}{N} \times \sum_{k=0}^{N-n-1} x(k)x(k+n)$$

where n is the period length (otherwise known as the lag value) (Giuliano Monti). If $x(n) = x(n+T)$ then the autocorrelation function $r(n)$ is also periodic with the same periodicity as the original function $x(n)$. Audio signals are very dynamic however and change drastically with time. Instruments exhibit different characteristics during their attack, sustain and decay phase (see Figure 3). The attack period being more percussive and less tonal, the sustain period being highly stationary with time and very tonal, and the decay period containing less tonal information and less power. This is why it is useful to use a short time autocorrelation function during the sustain phase, especially when the fundamental frequency is weak in power.

Peaks in the autocorrelation function correspond to strong periodic components. Because musical notes have harmonics that are at integer multiples of the fundamental frequency, a signal will exhibit high autocorrelation for multiples of the fundamental lag value. Because of this, the first significant value (height will give an indication of the periodicity of the signal) in the autocorrelation function after the zero lag value is the fundamental period (Giuliano Monti).

The benefit of using the autocorrelation function is that the computation is fast. The downside is that this method can require several periods of the signal to occur, and for low frequency fundamentals (down to C1 (32.75 Hz), this can take multiples of $1/32.75 = 30.5$ mili-seconds to occur, which puts a maximum on how long we have to wait to start analysing the data. Another common issue with using autocorrelation for pitch identification is that the algorithm may calculate a frequency that is twice the actual

fundamental frequency (“octave errors”).

- **FFT and First Peak Search** – The first part of this method involves sampling the incoming audio signal and storing the result into buffers. Somewhere between 1-61 milliseconds (enough for two periods of the lowest pitch expected) of data will be collected and stored. Then the Discrete Time Fourier Transform (DTFT) of the sampled data is taken using the following formula:

$$X_p(F) = \sum_{k=-\infty}^{\infty} x[k] \times e^{-j2\pi F}$$

Figure 4: DTFT

$$X(f) = \sum_{n=0}^{N-1} x(n) \sin\left(\frac{2\pi kn}{N}\right) + j \sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi kn}{N}\right)$$

Figure 5: FFT

$$x(n) = \sum_{n=0}^{N-1} X(f) \sin\left(\frac{2\pi kn}{N}\right) - j \sum_{n=0}^{N-1} X(f) \cos\left(\frac{2\pi kn}{N}\right)$$

Figure 6: IFFT

Where F is the digital frequency defined by the ratio of the analog frequency to the sampling rate and $x[k]$ is a digital sequence (Ambardar). A peak search algorithm would then begin where the first (lowest frequency) peak in the spectrum would be identified as the fundamental frequency. This process is slightly computationally expensive and so there are algorithms available to take Fast Fourier Transforms using certain numbers of samples, which require less computations per second.

- **Zero Crossing** – This method finds the fundamental frequency of a time varying waveform by finding the fundamental period. The points in time where there are zero crossings are measured and that information is used to find the fundamental period.

The pitch analysis methods presented here are not the only possible algorithms.

Requirements

Functional Requirements:

1. The system will use a microphone (dynamic or condenser) to capture an audio signal (of any length of time) coming from a recorder in real time.
2. An XLR cable from the microphone will be interfaced with a 3.5mm jack through an adapter. The signal from the microphone will be fed into the ADC (which must operate between 10 and 24 bits of dynamic range).
3. The system will detect what pitch is being played (fundamental frequency of incoming audio) and at what time using time domain analysis techniques and/or frequency domain analysis techniques.
4. The analysis will occur in real time. Calculations should be optimized and only run for as long as needed to identify a pitch.
5. The system will output MIDI data that corresponds to the notes in the original audio signal transcribed by the DSP engine, with a static velocity value of 80 (MIDI 1.0 Detailed Specification, 1995).

Performance Specifications:

1. The input audio signal must be monophonic (only one pitch occurs at a time), and come from a recorder (basic flute) playing at a SPL (sound pressure level) of at least 40 dB.
2. The analysis will occur in real time. Calculations should be optimized and only run for as long as needed to identify a pitch.
3. Less than 5% error rate for pitch identification. Less than 5% error for timing information extraction.
4. The output MIDI signal must be compatible with contemporary MDI to USB conversion technology.

Design Approaches

Approaches Considered

Ultimately, the choices made in terms of algorithm and platforms used came down to practical necessity. The platform used (the TMS320C5535 eZdsp board from Spectrum Digital and Texas Instruments) was the only one that we could afford, while appearing to have the capability to perform spectral processing in real time.

We chose the FFT and first peak search algorithm specifically because it was relatively simple, and the results we obtained using MATLAB met our specification.

We did not believe that using the zero crossing method would provide satisfactory results, however in retrospect, this method may have performed better. It certainly would take less

processing power.

MATLAB Results

The following plots were created in MATLAB using a recording of a recorder playing a Eb (MIDI note 79, fundamental frequency of 783.99 Hz) and then a F (MIDI note 81, fundamental frequency of 880 Hz).

To verify that the note timing information could accurately be deciphered, the following plot, figure 7, was created. The bottom figure plots the time domain waveform (where the note actually changed), and the top plot shows where the pitch recognition algorithm detected a change of pitch. The labels describe when our algorithm discovered the notes to change (the time of the change in the MIDI note), and where the notes actually changed (defined by a -3 dB drop in the amplitude of the audio signal). The window size was 22 ms – to obtain 1024 samples using a sample rate of 44.1 kHz. The computed note on and note off timing was within 5% of the actual note on and note off time. The error was 1.1% for note on, and 0.8% for note off)

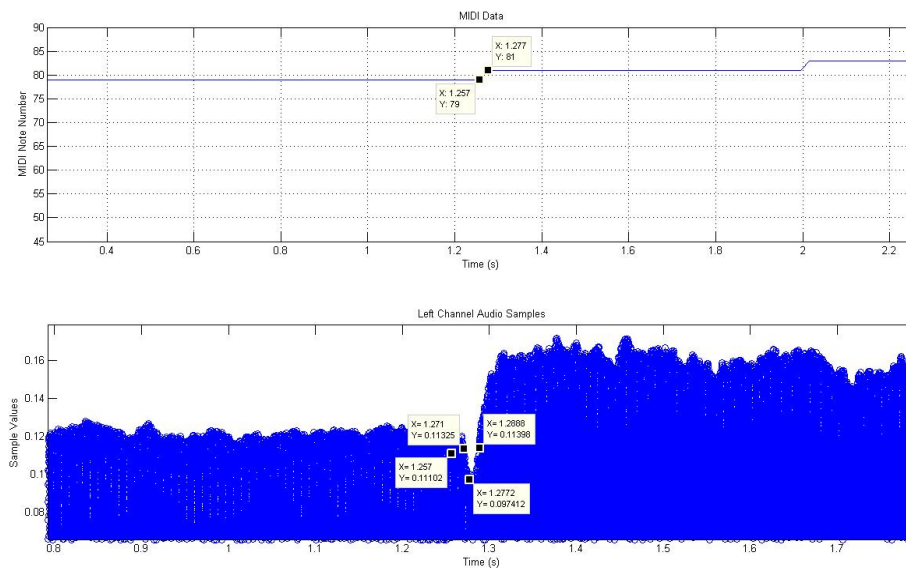


Figure 7: Results for the timing test for the FFT to First Peak Search algorithm.

To verify that the algorithm could correctly detect which pitches were present in the incoming audio signal, we used a recording of a recorder playing a Eb (MIDI note 79, fundamental frequency of 783.99 Hz). Using a 1024 point FFT, MATLAB found the peak to be 775 Hz (as can be seen in Figure 8), which is an error of 1.14%.

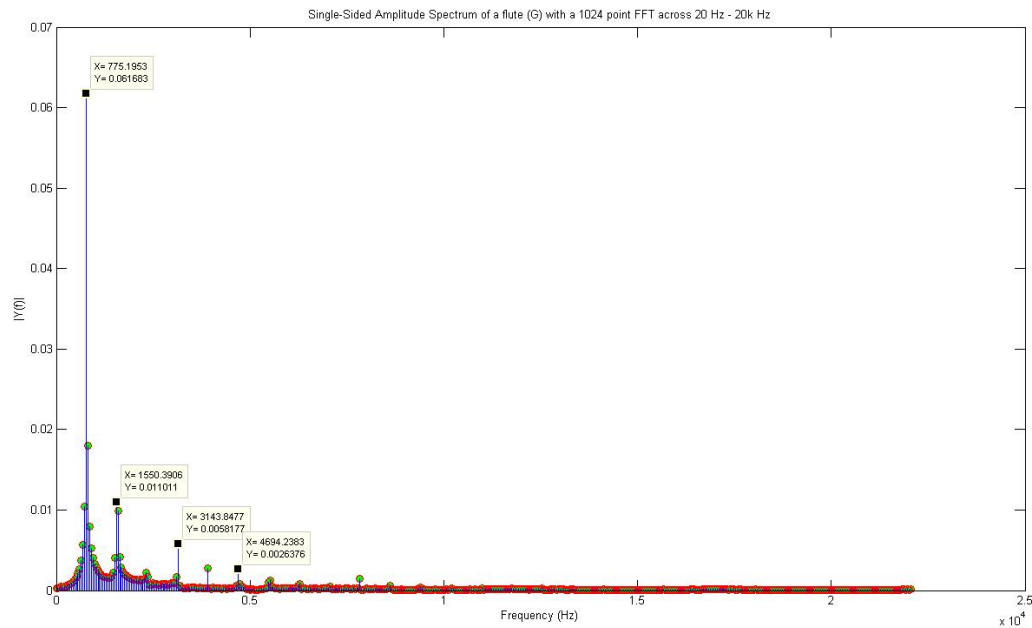


Figure 8: FFT of a recorder playing Eb, and the results of the peak search.

After this, a recording of a recorder playing MIDI note 79, 81, 83, 81, and 79 in succession was entered into MATLAB, and the results of the processing can be seen in figure 9 below.

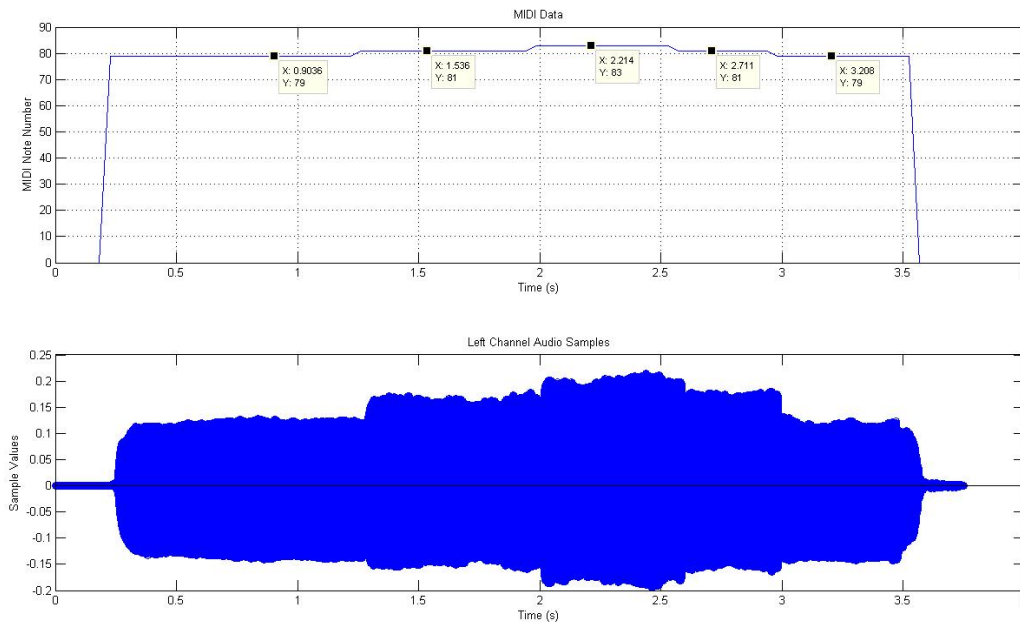


Figure 9: MATLAB pitch recognition results for a flute playing five contiguous notes.

As you can see, results in MATLAB show our algorithm to be successful in recognizing pitches.

Anti-aliasing filter topology

To create a proper anti-aliasing low pass filter, many filter topologies were simulated. Passive filters were not ideal due to loading, so we chose an active filter design. A Sallen-Key topology was used with a gain stage up front to compensate for inherent attenuation on the circuit. Circuit and simulation are shared below. Circuit topology can be seen in figure 10, and simulated circuit performance in LTspice in figure 11. Measured frequency response can be seen in figure 12.

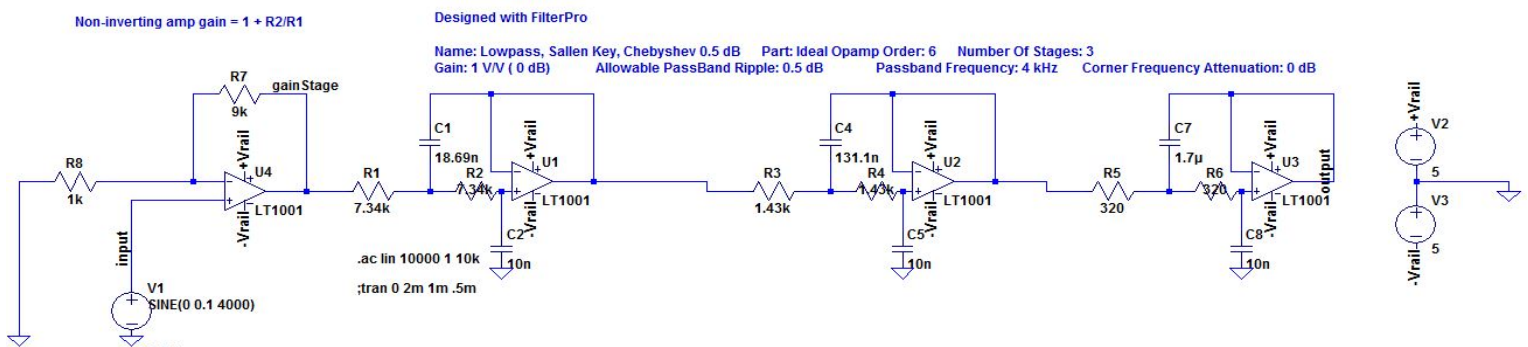


Figure 10: Anti-Aliasing schematic with a cutoff frequency of 4 kHz.

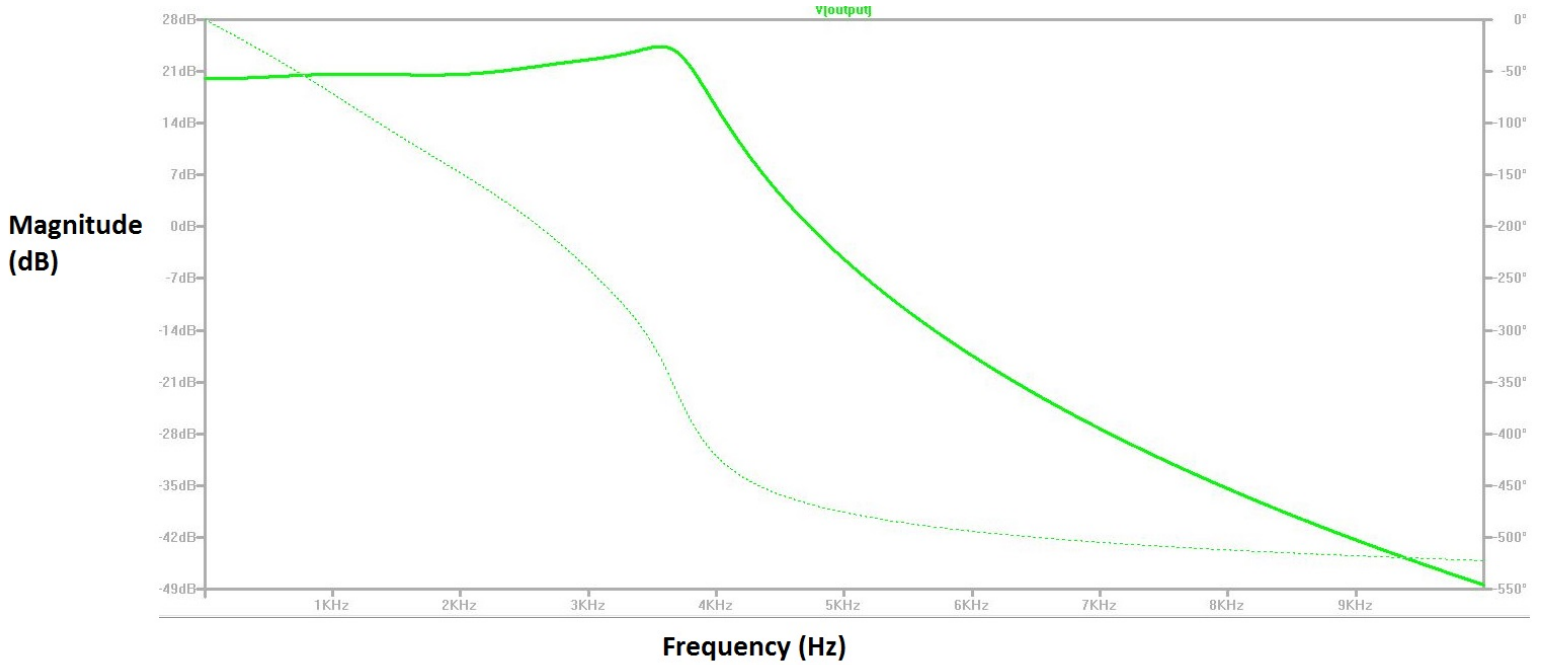


Figure 11: Anti-Aliasing filter simulated frequency response.

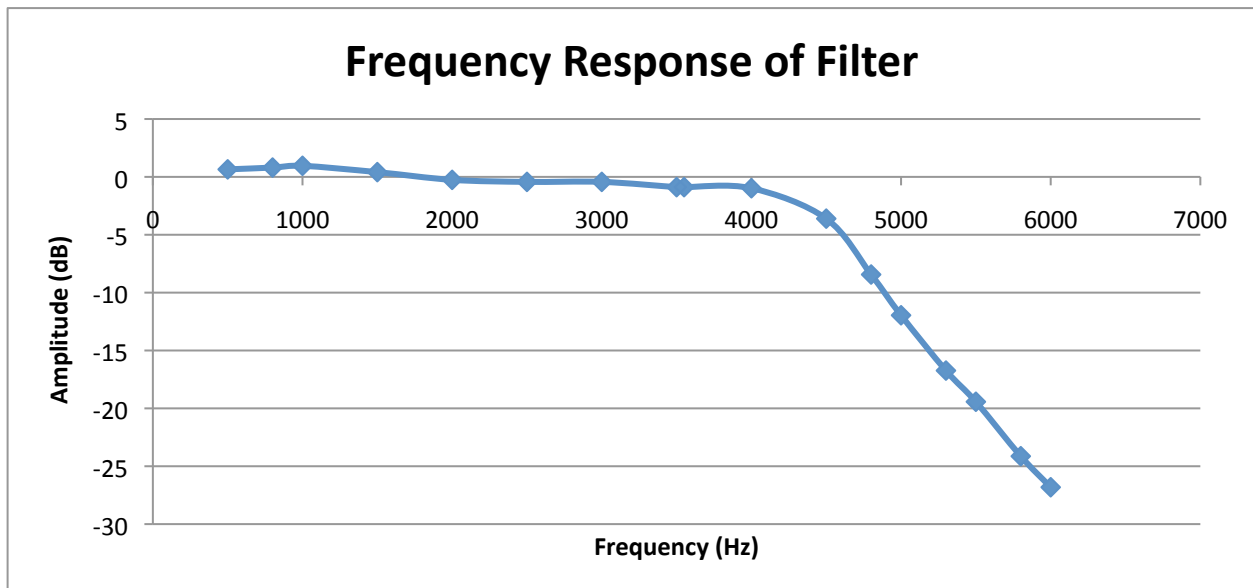


Figure 12: Measured frequency response of filter.

Project Design

User Interface

Figure 13 describes a summary of the system.



Figure 13: A high level representational diagram for the system.

The output stage is a standard MIDI connector which the user can use with any MIDI device. The two inverters and the resistor act as an impedance buffer and bring the voltage level to the appropriate range for MIDI.

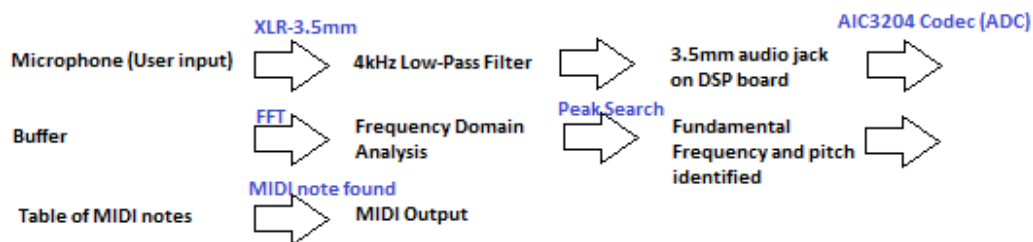


Figure 14: Black box diagram of the system.

Figure 14 represents a signal flow graph of the system.

A detailed block diagram for the signal flow, and algorithm combined can be seen in figure 15.

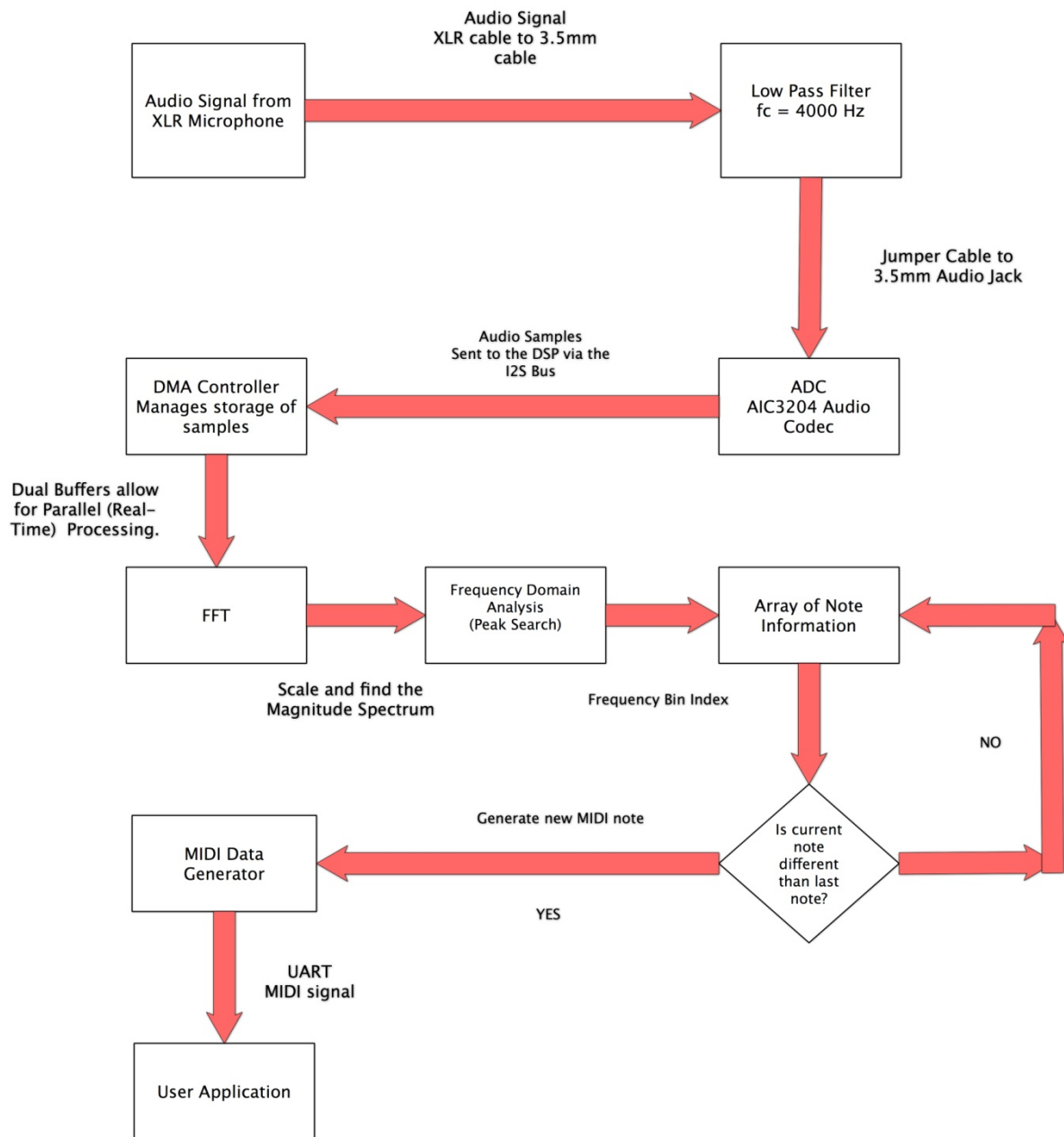


Figure 15: Complete system block diagram.

Description of Subsystems and Theory of Operation

- Filter : This 4kHz low-pass filter is meant to prevent aliasing. The fundamentals that we are concerned with appear below 4kHz, so we do not want to bother with any other frequencies. We have found that the filter does not noticeably affect system performance.
- The AIC3204 Audio Codec, which is initialized by the CPU using I2C, samples at a rate of 8 kHz, using 16 bit samples, which are transferred via an I2S bus to a DMA controller contained inside the C5535 chip.
- The DMA controller fills two buffers successively (ping and pong buffers) that are each 1024 elements wide. This allows the CPU to process one array while the other is being filled with new audio data.
- The CPU uses a radix-2 implementation to compute the FFT. The results are scaled appropriately to prevent against arithmetic overflow. The magnitude spectrum is then found.
- A peak search is then performed on the resulting spectrum to find the frequency bin where the most energy is contained in (which will usually be the fundamental frequency for most instruments, including the recorder).
- Once the frequency bin which contains the most energy is identified, a Look-Up-Table is used to match the frequency bin to the appropriate MIDI note number.
- Pitch information is output in standard MIDI format (using a UART within the C5535 chip) to be used with any MIDI device/software.

Physical Construction and Integration

There is no physical enclosure needed for the device. The DSP evaluation board plugs into a computer via a USB 2.0 port. The sound source is connected to the anti-aliasing filter using a 3.5mm jack. The output of the filter connects to the DSP via a 3.5mm cable. The output MIDI signal is connected to the MIDI interfacing circuit using jumper wire. The output of the MIDI interfacing circuit is connected to a standard 5 pin DIN connector.



Figure 16: Physical Implementation. Top left audio jack is microphone input. Signal runs through filter and is input to audio jack on DSP board. After processing this data, MIDI output is sent through connector pin to MIDI interfacing hardware.

Integrated System Tests and Results

The original specifications were not met in their entirety. The functional requirements were met as follows:

1. The system uses a dynamic microphone to capture an audio signal (of any length of time). The input audio signal must be monophonic (only one pitch occurs at a time), and come from a recorder (basic flute) playing at a SPL (sound pressure level) of at least 40 dB.
2. An XLR cable from the microphone is interfaced with a 3.5mm jack through an adapter. The signal from the microphone is fed into an onboard ADC which operates with 10 bits of dynamic range.

3. The system attempts to detect what pitch is being played (fundamental frequency of incoming audio) and at what time it is being played using frequency domain analysis.
4. The system will output MIDI data that corresponds to the notes in the original audio signal transcribed by the DSP engine, with a static velocity value of 80 (MIDI 1.0 Detailed Specification, 1995).

The performance specifications were met as follows:

1. The analysis occurs in real time.
2. **Not met** - Less than 5% error rate for pitch identification. Less than 5% error for timing information extraction.
3. The output MIDI signal is compatible with contemporary MIDI to USB conversion technology.

Testing involved inputting a .wav recording of a flute playing 5 notes in succession. These notes were MIDI note #'s 84, 86, 88, 86, 84. The following table summarizes the results generated from our algorithm on the dsp board. The .wav file was input to the system 3 separate times, as each separate instance produced different results. The worst case error is reported in table 2.

Actual Note (MIDI #)	Actual t_{ON} (ms)	Actual t_{OFF} (ms)	Max Error for Generated Note	Generated t_{ON} (ms)	Generated t_{OFF} (ms)
84	526	1253	12	530	1267
86	1288	2112	12	1292	2119
88	2145	2745	6	2155	2754
86	2767	3649	12	2775	3661
84	3687	4582	12	3690	4588

Table 1: Summary of test results for the trial involving 5 contiguous flute notes.

	Max Error (%)
Note	14.200%
t_{ON}	0.754%
t_{OFF}	1.104%

Table 2: Maximum error from the three test cases involving five contiguous flute notes.

As can be seen in table 1 and 2, the requirements for the pitch identification were not met, but the requirements for the note on and off timing information were met.

See figure 20 for a graphical representation of the third trial in which the .wav file was input into the system. The yellow marking represent MIDI information being displayed using Apple's Logic Pro. Figure 19 shows how the MIDI information should be displayed, if the system operated with no error.

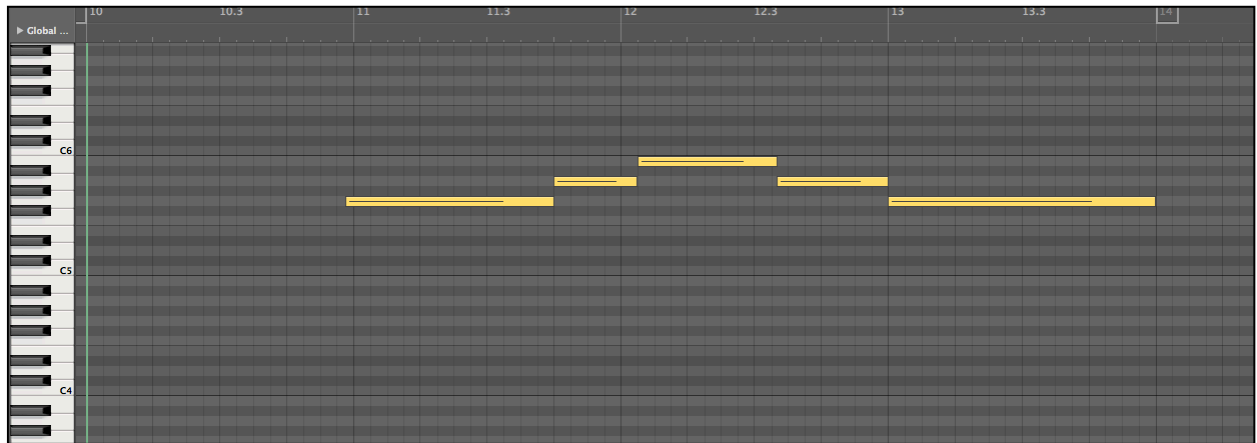


Figure 17: Expected results from the third test trial involving a recording of a flute playing MIDI notes 84, 86, and 88.

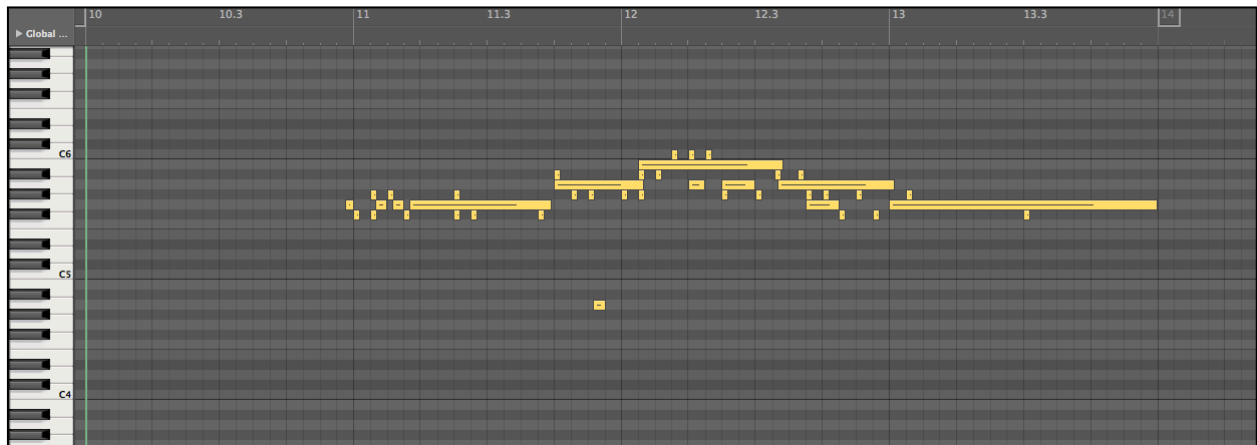


Figure 18: Actual experimental results of the third test trial. The longer sustained notes are correct, but the small MIDI notes are incorrect pitches being identified.

The problem with our peak detection system is that the DSP locates peaks in the signal energy in frequency bins surrounding the frequency bin in which we expect the peak to appear. We tested this by inputting a pure 1000 Hz sinusoid, and found that the DSP found peaks anywhere from

970 to 1040 Hz on average, and sometimes even found peaks hundreds of Hertz away from 1000 Hz.

Conclusions

The original specifications were not met entirely. The functional requirements of the system, however, were met. The proper inputs and outputs were functional for the format that we specified. The performance specifications were not met. Since the system could not properly identify the fundamental frequency even when a pure sinusoid was input, it was doubtful that the system could correctly identify the fundamental when a more complicated signal (a recorder) was input. Indeed, the system could *not* consistently identify the correct pitch recorder was played into the microphone. Peaks in the energy were clearly found in frequency bins around the frequency bin in which we expected the peak to appear. See figure 17 for a tabulated summary of the error in our trials involving the pre-recorded flute.

Unfortunately because the TMS320C5535 eZdsp board had just been released, documentation, support and resources for using the chip and its peripherals were scarce. Existing documentation often had errors, and contradictions. The majority of this project was spent understanding how to get the chip to correctly communicate with all of the peripherals (I2C, I2S, the ADC, DMA, and UART). Once the chip was successfully receiving audio data, processing it, and transmitting MIDI data, there was little time to troubleshoot and resolve the large margin of error we were seeing.

If we had more time to solve this problem, we would compare FFT results generated from MATLAB with FFT results generated from the DSP - using the same source. The FFT computation was done using libraries provided from Texas Instruments, which means that the FFT is most likely being computed correctly. If MATLAB and the DSP provided comparable results, we would then proceed to examine the processing that is done to the frequency spectrum, however because our algorithm is so simple, it is likely that no problem exists in that portion of the source code. The MIDI generation portion of the code as well as the procurement of audio samples was tested to be working correctly.

Our system has the necessary frequency resolution for pitch detection in the range of a recorder flute, but increasing the frequency resolution even more may be of help. Of course, if more FFT points are used as a window, the time domain resolution will suffer, however because the samples are taken at a rate of 44.1 kHz or 48 kHz, it is unlikely that time domain resolution will be an issue, as using a 1024 point FFT uses a time domain window of around 11 ms (assuming 44.1 kHz sampling rate), which is more than enough time resolution to resolve the fastest notes that the vast majority of potential users would play.

If we were to do this again, we would look into using a different platform. There exist DSP chips that software on computers allow you to program graphically. This could have made programming and using peripherals much easier, which would have allowed more time to troubleshoot. These other DSPs may have more resources and a larger community using them as well.

Overall, the project introduced us to the world of DSPs, and we have learned an incredible amount in attempting to develop the Audio to MIDI converter. The project provided a chance to learn C, microprocessor architecture, digital data communication protocols, analog filters, MATLAB, and DSP theory, among other skills.

Thanks for reading!

Bibliography

Ambardar, A. (1999). *Analog and Digital Signal Processing*. Independence, Kentucky: Brooks Cole.

C5535 eZdsp USB Stick Development Kit. (n.d.). Retrieved 2012 18-November from Texas Instruments: <http://www.ti.com/tool/tmdx5535ezdsp#Technical%20Documents>

Chapman, D. (n.d.). *The Sound of the African Thumb Piano*. Retrieved 2012 18-November from acoustics.org: <http://www.acoustics.org/press/155th/chapman.htm>

Corbet, C. (n.d.). *MATLAB EQ*. Retrieved 2012 18-November from cnx.org: <http://cnx.org/content/m15655/latest/>

Coutant. (n.d.). Retrieved 2012 19-November from <http://www.coutant.org/xlr.html>

Ekstrom, L. (2010 6-January). *Allo on MIDI*. Retrieved 2012 18-November from alodk: http://alodk.dk/alomidi_uk.htm

Feilding, C. (n.d.). *Hearing VII*. Retrieved 2012 18-November from College of Santa Fe Auditory Hearing: http://www.feilding.net/sfuad/musi3012-01/html/lectures/012_hearing_VII.htm

Fitz, K. (2003 November). *Current Research in Real-time Sound Morphing*. Retrieved 2012 18-November from hakenaudio.com: <http://www.hakenaudio.com/RealTimeMorph/>

Giuliano Monti, M. S. (2000). *Monophonic Transcription with Autocorrelation*. 4.

Mall, 6. (n.d.). Retrieved 2012 19-November from 61ic Mall: <http://www.61ic.com.cn/product-125.htm>

MIDI 1.0 Detailed Specification (4.2 ed.). (1995). Los Angeles, CA: MIDI Manufacturers Association.

Northrup, H. (n.d.). *Howard's Musical Instrument & Recording Equipment Photo Album*. Retrieved 2012 19-November from Howard Northrup: <http://www.howardnorthrup.com/photos/index.album/shure-sm58-microphone?i=6&s=1>

Pitch Detection Methods. (2012 12-November). Retrieved 2012 18-November from http://sound.eti.pg.gda.pl/student/eim/synteza/leszczyna/index_ang.htm

Terminology. (n.d.). Retrieved 2012 18-November from Minelab: <http://www.minelab.com/usa/consumer/knowledge-base/terminology>

Appendices

Specifications

1. The system uses a dynamic microphone to capture an audio signal (of any length of time). The input audio signal must be monophonic (only one pitch occurs at a time), and come from a recorder (basic flute) playing at a SPL (sound pressure level) of at least 40 dB.
2. An XLR cable from the microphone is interfaced with a 3.5mm jack through an adapter. The signal from the microphone is fed into an onboard ADC which operates with 10 bits of dynamic range.
3. The system attempts to detect what pitch is being played (fundamental frequency of incoming audio) and at what time it is being played using frequency domain analysis.
4. The system will output MIDI data that corresponds to the notes in the original audio signal transcribed by the DSP engine, with a static velocity value of 80 (MIDI 1.0 Detailed Specification, 1995).
5. The analysis occurs in real time.
6. **Not met** - Less than 5% error rate for pitch identification. Less than 5% error for timing information extraction.
7. The output MIDI signal is compatible with contemporary MIDI to USB conversion technology.

Parts List and Costs

Below is the minimum BOM for a working project. We bought more parts as extras.

Basic materials for circuit building, such as a breadboard and wires, are not included.

Cost does not include tax or shipping & handling.

Item	Quantity	Unit Cost (\$)
C5535 eZdsp USB Stick Dev Kit	1	100
3.5mm to 3.5mm M/M	2	7.50
XLR – 3.5mm	1	10.50
Expansion Pin Connector	1	7.48
Filter Capacitors	6	About 0.5
Trim Resistors for Filter	6	About 0.5
LM 741 Op-Amp	3	About 1.00
MIDI Connector – Female Right Angle	1	6.00
Power Supply ($\pm 5V$)	1	Provided
Total	N/A	247.98

Schedule

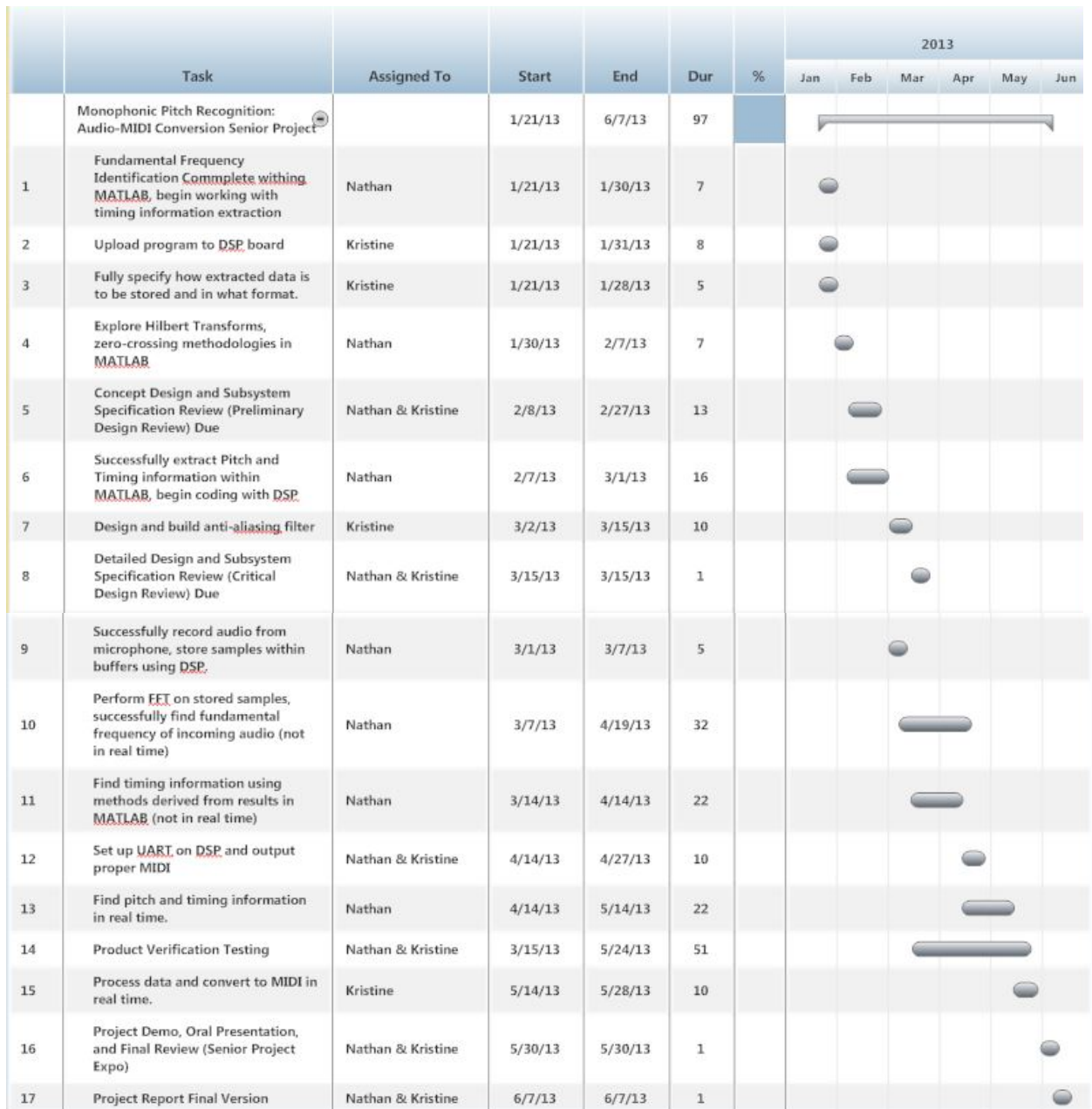


Figure 19: Work schedule for the project.

Program Listing

The following program listing includes most of the code used in the program – but excludes all listed header files.

```
Main.c
/* Author: Nathan Zorndorf
 * Description: Audio to MIDI converter.
 */

#include <stdio.h>
#include <math.h>
#include <tms320.h>
#include <Dsplib.h>
#include <my_types.h>
#include <usbstk5505.h>
#include <Application_1_Modified_Registers.h>
#include <My_PLL.h>
#include <My_I2C.h>
#include <My_AIC3204.h>
#include <My_I2S_Register.h>
#include <My_DMA_Ping_Pong_Register_Setup.h>
#include <My_UART.h>
#include <Audio_To_MIDI_Using_DMA_and_CFFT.h>

// Buffers for DMA in/out and overlap-add
Int16 PingPongFlagInL;
Int16 PingPongFlagInR;
Int16 PingPongFlagOutL;
Int16 PingPongFlagOutR;

Int16 DMA_InpL[PING_PONG_SIZE];
Int16 DMA_InpR[PING_PONG_SIZE];
Int16 DMA_OutL[PING_PONG_SIZE];
Int16 DMA_OutR[PING_PONG_SIZE];

int main(void) {

    printf("Hello Nathan. Let's get funky. \n");

    My_PLL();

    IER0 = 0xC100; // DMA, I2S RX/TX Interrupt Enable.

    My_I2C();

    My_AIC3204();

    My_DMA_Ping_Pong_Register_Setup();
```

```

    My_UART();

    My_I2S_Register();

    Audio_To_MIDI_Using_DMA_and_CFFT();

    printf("Reached end of main!\n");

    return(1);
}

```

My_PLL.c

```

/*
=====
=====
*   Author: Nathan Zorndorf
*   Description: Setting up the PLL
*/

#include <stdio.h>
#include <my_types.h>
#include <usbstk5505.h>
#include <aic3204.h>
#include <PLL.h>
#include <stereo.h>
#include <usbstk5505_gpio.h>
#include <usbstk5505_i2c.h>
#include <csl_general.h>
#include <csl_pll.h>
#include <csl_pllAux.h>
#include <Application_1_Modified_Registers.h>

#define PLL_CNTL1      *((ioport volatile unsigned *)0x1C20    // PLL
Control Register #1
#define PLL_CNTL2      *((ioport volatile unsigned *)0x1C21
#define PLL_CNTL3      *((ioport volatile unsigned *)0x1C22
#define PLL_CNTL4      *((ioport volatile unsigned *)0x1C23

int My_PLL(void) {

    Uint16  register_value;
    Uint16  i=0;
    //----- PLL SETUP BEGIN -----
    printf("PLL SETUP BEGIN! \n");

    // Enable clocks to all peripherals
    SYS_PCGCR1 = 0x0000;

```

```

SYS_PCGCR2 = 0x0000;

// Set the UART CLKSTOP bits to 0 to allow the clock to the UART
peripheral.
register_value = CPU_CLKSTOP;
CPU_CLKSTOP = register_value & 0xFFCF;

// External Bus Selection Register
CPU_EBSR = 0x1000; // Mode 1 (SPI, GPIO, UART, and I2S2) - OK

/* CGCR1 - Section 1.4.4.1 */
/* PLL power up. PLL Multiplier M = 1000 */
PLL_CNTL1 = 0x8BE8; //PG1.4: 0x82FA;
/* CGCR2 - Section 1.4.4.2 */
PLL_CNTL2 = 0x8000; /* Bypass reference divider */
/* CGCR3 - Section 1.4.4.3 */
PLL_CNTL3 = 0x0806; /* initialization bits */
/* CGCR4 - Section 1.4.4.4 */
PLL_CNTL4 = 0x0000; /* Bypass output divider */

// Peripheral Reset
CPU_PSRCR = 0x0008; // Software reset signals asserted after 2
clock cycles
CPU_PRCR = 0x00FF; // Reset all peripherals.
for (i=0; i< 0xFFFF; i++); // Wait in for loop to give time for
peripherals to reset.

//----- PLL SETUP END -----
printf("PLL SETUP END \n");
printf("REGISTER          --- CONFIG VALUES\n");
printf("PLL_CGCR1            --- 0x%X \n", PLL_CGCR1);
printf("PLL_CGCR2            --- 0x%X \n", PLL_CGCR2);
printf("PLL_CGCR3            --- 0x%X \n", PLL_CGCR3);
printf("PLL_CGCR4            --- 0x%X \n", PLL_CGCR4);
printf("PLL_CCR2             --- 0x%X \n", PLL_CCR2);

return(0);
}

```

My_I2C.c

```

/* Author: Nathan Zorndorf
 * Description: Testing and using the I2C to program the AIC3204
 */

#include <stdio.h>
#include <my_types.h>
#include <usbstk5505.h>
#include <aic3204.h>

```

```

#include <PLL.h>
#include <stereo.h>
#include <usbstk5505_gpio.h>
#include <usbstk5505_i2c.h>
#include <csl_general.h>
#include <csl_pll.h>
#include <csl_pllAux.h>
#include <csl_i2c.h>
#include <csl_i2s.h>
#include <aic_i2c.h>

int My_I2C(void) {

    // ----- I2C Setup -----
    printf("I2C SETUP BEGIN!!\n");

    I2C_MDR = 0x0400;           // Reset I2C
    I2C_PSC  = 7;               // Config prescaler for 12MHz
    I2C_CLKL = 0x026C;          // Config clk LOW for 20kHz
    I2C_CLKH = 0x026C;          // Config clk HIGH for 20kHz

    I2C_MDR  = 0x0420 ;         // Release from reset; Master,
    Transmitter, 7-bit address

    printf("I2C SETUP END!!\n\n");
    // -----

    return(0);

}

```

My_AIC3204.c

```

/*
=====
=====
*   Author: Nathan Zorndorf
*   Description: Setting up AIC3204 Audio Codec
*/

#include <stdio.h>
#include <my_types.h>
#include <usbstk5505.h>
#include <aic3204.h>
#include <PLL.h>
#include <stereo.h>
#include <usbstk5505_gpio.h>
#include <usbstk5505_i2c.h>

```

```

#include <csl_general.h>
#include <csl_pll.h>
#include <csl_pllAux.h>
#include <csl_i2c.h>
#include <csl_i2s.h>
#include <aic_i2c.h>
#include <My_AIC3204.h>

int My_AIC3204(void) {

    /* Define sampling rate parameters */
    unsigned char pll_dh, pll_dl, madc, mdac, nadc, ndac;
    /* Choose AIC3204 sampling rate */
    short fs = AIC3204_FS_8KHZ;

    /* Set Sampling Rate */
    if(fs == AIC3204_FS_44_1KHZ)
    {
        // PLL D = 560
        pll_dh = 0x02;
        pll_dl = 0x30;

        // MDAC & MACC = 3
        madc = 0x03;
        mdac = 0x03;

        // NDAC & NADC = 5
        nadc = 0x05;
        ndac = 0x05;
    }
    else // 8, 16, 24, 32, 48, 96 KHz
    {
        // PLL D = 1680
        pll_dh = 0x06;
        pll_dl = 0x90;

        switch(fs) {
            case AIC3204_FS_8KHZ:
                madc = 0x0C;
                mdac = 0x0C;
                break;
            case AIC3204_FS_16KHZ:
                madc = 0x06;
                mdac = 0x06;
                break;
            case AIC3204_FS_24KHZ:
                madc = 0x04;
                mdac = 0x04;
                break;
            case AIC3204_FS_32KHZ:

```

```

        madc = 0x03;
        mdac = 0x03;
        break;
    case AIC3204_FS_48KHZ:
        madc = 0x02;
        mdac = 0x02;
        break;
    case AIC3204_FS_96KHZ:
        madc = 0x01;
        mdac = 0x01;
        break;
}

// NDAC & NADC = 7
nadc = 0x07;
ndac = 0x07;
}

/* these variables declared for reading AIC3204 register values */
//Uint16 test_reg_val;
//int i;

printf("AIC3204 SETUP BEGIN\n");
// Configure Parallel Port
SYS_EXBUSSEL |= 0x1000; // Configure Parallel Port mode = 1 for
I2S2 (SPI, GPIO, UART, and I2S2).
// Define Starting Point - OK
AIC3204_rset( 0, 0); // Select PAGE 0 - OK
AIC3204_rset(0x01, 0x01); // Software Reset codec - OK
// Clock Settings - ?
printf("PLL and Clocks config and Power Up...\n");
AIC3204_rset( 0, 0); // Select PAGE 0 - OK
AIC3204_rset(0x1B, 0x0D); // Audio Interface Setting Register 1
(P0_R27) - OK
// I2S, 16bit word length, BCLK and WCLK is set as o/p from
AIC3204, DOUT will be high impedance after data has been transferred
AIC3204_rset(0x1C, 0x00); // Data offset = 0 - OK
AIC3204_rset(0x04, 0x43); // MCLK = PLL_CLKIN => *R*J.D/p =>
PLL_CLK => PLLCLK => CODEC_CLKIN and HIGH PLL clock range - OK
AIC3204_rset(0x05, 0x91); // Power up PLL, P=1, R=1 - OK
AIC3204_rset(0x06, 0x07); // J = 7 - OK
AIC3204_rset(0x07, pll_dh); // D (MSBits) D=1680 - OK
AIC3204_rset(0x08, pll_dl); // D (LSBits) - OK
AIC3204_rset(0x1E, 0x88); // WCLK source, BCLK Power UP/down, and
BCLK divider value = 8 - BCLK is output to I2S2_CLK on CPU
// BCLK=DAC_CLK/N =(12288000/8) =
1.536MHz, which must be > 32*(FS/WCLK/ADC_FS) because that way
// 32 bits can be transferred per
sample
// this setup -> DAC_CLK = WCLK, and

```



```

BCLK = DAC_CLK/BCLKDividervalue
AIC3204_rset(0x0B, 0x80 | ndac);      // NDAC = 7    - OK
AIC3204_rset(0x0C, 0x80 | mdac);      // MDAC = 2    - OK
AIC3204_rset(0x0D, 0x00);              // DOSR(MSbits) = 0
    - OK
AIC3204_rset(0x0E, 0x80);              // DOSR(LSbits) = 128 decimal or
0x0080 - OK
AIC3204_rset(0x12, 0x80 | nadc);      // NADC = 7    - OK
AIC3204_rset(0x13, 0x80 | madc);      // MADC = 2    - OK
AIC3204_rset(0x14, 0x80);              // AOSR = 128 - OK
AIC3204_rset(0x3D, 0x01);              // ADC PRB_R1 - OK
// Program Analog Blocks    - OK
printf("Program Analog Blocks...\n");
AIC3204_rset( 0, 1);                  // Select PAGE 1    - OK
AIC3204_rset(0x01, 0x08);              // Disable Internal Crude AVdd - OK
AIC3204_rset(0x02, 0x01);              // Enable Master Analog Power
// DAC Routing and Power Up - OK
printf("DAC ROUTING and Power Up... \n");
AIC3204_rset( 0, 1);                  //Select PAGE 1    - OK
AIC3204_rset(0x0C, 0x08);              //Left Channel DAC reconstruction
filter's positive terminal is routed to HPL - OK
AIC3204_rset(0x0D, 0x08);              //Right Channel DAC reconstruction
filter's positive terminal is routed to HPL - OK
AIC3204_rset( 0, 0);                  //Select PAGE 0    - OK
AIC3204_rset(0x40, 0x02);              //DAC unmute, right volume tracks
left - OK
AIC3204_rset(0x41, 0x00);              //DAC Left Volume = 0dB - OK
AIC3204_rset(0x3F, 0xD4);              // - OK
                                        //R&L DAC data = R&L
Channel Audio Interface Data, Soft-Stepping is 1 step per 1 DAC Word
Clock, R&L DAC Power Up
AIC3204_rset( 0, 1);                  // Select PAGE 1    - OK
AIC3204_rset(0x10, 0x3A );            // Unmute HPL , -6 dB gain- OK
AIC3204_rset(0x11, 0x3A );            // Unmute HPR , -6 dB gain- OK
AIC3204_rset(0x09, 0x30);              // Power up HPL,HPR    - OK
// ADC Routing
    - OK
printf("ADC Routing...\n");
AIC3204_rset( 0, 1 );                  //Select PAGE 1    - OK
AIC3204_rset(0x33, 0x00);              // MICBIAS power down - MCBIAS is
for "electret-condenser microphones," we are using a dynamic mic - OK
AIC3204_rset(0x34, 0x30);              // IN2L routed to Left MICPGA
(positive terminal) via 40k0hm = -12 dB gain    - OK
AIC3204_rset(0x37, 0x30);              // IN2R routed to Right MICPGA
(positive terminal)via 40k0hm = -12 dB gain - OK
AIC3204_rset(0x36, 0x03);              // CM2L routed to left MICPGA
(negative terminal) via 40k0hm    - OK
AIC3204_rset(0x39, 0x03);              // CM2R routed to right MICPGA
(negative terminal)via 40k0hm    - OK
AIC3204_rset(0x3B, 45);                // MICPGA (left) enabled, register 59,
value 0x28(40d) = +20 dB gain    - OK

```

```

    AIC3204_rset(0x3C, 45);    // MICPGA (right) enabled, register 60,
    value 0x28(40d) = +20 dB gain    - OK
    // ADC Power Up                - OK
    printf("ADC Power Up...\n");
    AIC3204_rset( 0, 0 );      // Select page 0
                                - OK
    AIC3204_rset(0x51, 0xC0);   // Left and Right channel ADC power
up                               - OK
    AIC3204_rset(0x52, 0x00);   // Left and Right ADC unmuted, fine
gain on both channels = 0dB- OK

    USBSTK5505_wait( 100 );    // Wait
    printf("AIC3204 SETUP END\n\n");

    return(0);
}

```

My_DMA_Ping_Pong_Register_Setup.c

```

/*
=====
=====
*   Author: Nathan Zorndorf
*   Description: Setting up the DMA in Ping Pong mode by writing
directly to registers.
*/

#include <soc.h>
#include <csl_intc.h>
#include <stdio.h>
#include <usbstk5505.h>
#include <My_I2S_Register.h>
#include <Application_1_Modified_Registers.h>
#include <My_DMA_Ping_Pong_Register_Setup.h>

/* Interrupt vector start address */
extern void VECSTART(void);           // WHERE IS THIS
DEFINED/DECLARED

int My_DMA_Ping_Pong_Register_Setup(void) {

    Uint16 register_value;
    Uint32 dma_address;

    // ----- Interrupts Setup ----- /
    printf("DMA INTERRUPTS SETUP BEGIN!!\n");
    //IRQ_globalDisable(); // It disables the interrupt globally by
disabling INTM bit and also return the previous mask value for INTM

```

```

bit
    IRQ_clearAll(); // This function clears all the interrupts. Both
    IFR0 and IFR1 are cleared by this function.
    //IRQ_disableAll(); // This function disables all the interrupts
    available on C5505 DSP. Both IER0 and IER1 are cleared by this function
    IRQ_setVecs((Uint32)(&VECSTART)); // It stores the Interrupt
    vector table address in Interrupt vector pointer DSP and Interrupt
    vector pointer host Registers
    //IRQ_clear(DMA_EVENT); // possibly not necessary and
    might need to go before(?) or after(?) IRQ_plug()
    IRQ_plug(DMA_EVENT, &DMA_ISR); // This function is used to
    register the ISR routine for the corresponding interrupt event.
    IRQ_enable(DMA_EVENT); // It enables the corresponding interrupt
    bit in IER Register and also return the previous bit mask value
    IRQ_globalEnable(); // It enables the interrupt globally by
    enabling INTM bit and also return the previous mask value for INTM bit
    printf("DMA INTERRUPTS SETUP END!!\n");
    // ----- /

printf("DMA SETUP BEGIN!!\n");

// step 3
DMA_IFR = 0xFFFF; // Enable Interrupts for DMA 1, Channels 0-3
IFR0 = 0x0100; // Clear DMA CPU interrupt flag
printf("DMA_IFR = 0x%X \n", DMA_IFR); // 1 = DMA controller
n, channel m block transfer complete.
printf("IER0 = 0x%X \n", IER0);
printf("IER1 = 0x%X \n", IER1);
printf("IER0 = 0x%X \n", IFR0);
printf("IER1 = 0x%X \n", IFR1);

// Step 4 - Enable interrupts
DMA_IER = 0x00F0; // Enable interrupts for DMA1 CH0-CH3

// Step 5 - Setup sync event
register_value = DMA1_CESR1;
DMA1_CESR1 = register_value | 0x0202; /* Set CH1, CH0 sync
event to I2S2 receive */
register_value = DMA1_CESR2;
DMA1_CESR2 = register_value | 0x0101; /* Set CH3, CH2 sync
event to I2S2 transmit*/

// Step 6 - Channel Source Address
DMA1_CH0_SSAL = 0x2A28; // I2S Receive Left Data 0 Register
DMA1_CH0_SSAU = 0x0000;

DMA1_CH1_SSAL = 0x2A2C; // I2S Receive Right Data 0 Register
DMA1_CH1_SSAU = 0x0000;

```

```

    dma_address = convert_address(DMA_OutL);           // convert address
    DMA1_CH2_SSAL = (Uint16)dma_address;               // keep LSBs
    DMA1_CH2_SSAU = 0xFFFF & (dma_address >> 16);    // keep MSBs

    dma_address = convert_address(DMA_OutR);
    DMA1_CH3_SSAL = (Uint16)dma_address;
    DMA1_CH3_SSAU = 0xFFFF & (dma_address >> 16);

    // Step 7 - Channel Destination Address
    dma_address = convert_address(DMA_InpL);
    DMA1_CH0_DSAL = (Uint16)dma_address;
    DMA1_CH0_DSAU = 0xFFFF & (dma_address >> 16);

    dma_address = convert_address(DMA_InpR);
    DMA1_CH1_DSAL = (Uint16)dma_address;
    DMA1_CH1_DSAU = 0xFFFF & (dma_address >> 16);

    DMA1_CH2_DSAL = 0x2A08; // I2S2 Transmit Left Data 0 Register
    DMA1_CH2_DSAU = 0x0000;

    DMA1_CH3_DSAL = 0x2A0C; // I2S2 Transmit Right Data 0 Register
    DMA1_CH3_DSAU = 0x0000;

    /* Step 8 - DMA Transfer Length */
    /* Note the following (Sections 2.9, 4.3):
    */
    /* - DMA transfer length is specified in bytes.
    */
    /* - DMA transfer length in ping/pong mode is half the length of
    */
    /*   the TCR1 register.
    */
    /* - Ping/pong buffers should occupy contiguous memory spaces and
    */
    /*   it is recommended to setup a buffer double the size of the
    */
    /*   intended one to ensure this.
    */
    /* In order to transfer _AUDIO_IO_SIZE_ 16-bit samples in ping/pong
    */
    /* mode, we specify the transfer length as 2*2*AUDIO_IO_SIZE =
    2*PING_PONG_SIZE */
    DMA1_CH0_TCR1 = 2*PING_PONG_SIZE;
    DMA1_CH1_TCR1 = 2*PING_PONG_SIZE;
    DMA1_CH2_TCR1 = 2*PING_PONG_SIZE;
    DMA1_CH3_TCR1 = 2*PING_PONG_SIZE;

    // Step 9 - Configure options
    DMA1_CH0_TCR2 = 0x3081; // source is constant, destination address
    increments by four bytes after each transfer.

```

```

DMA1_CH1_TCR2 = 0x3081;
DMA1_CH2_TCR2 = 0x3201; // destination is constant, source address
increments by four bytes after each transfer.
DMA1_CH3_TCR2 = 0x3201;

// Step 10 - Enable DMA Controller 0 channel 0-3
register_value = DMA1_CH0_TCR2;
DMA1_CH0_TCR2 = register_value | 0x8004;
register_value = DMA1_CH1_TCR2;
DMA1_CH1_TCR2 = register_value | 0x8004;
register_value = DMA1_CH2_TCR2;
DMA1_CH2_TCR2 = register_value | 0x8004;
register_value = DMA1_CH3_TCR2;
DMA1_CH3_TCR2 = register_value | 0x8004;

return(0);
}

interrupt void DMA_ISR(void)
{
    Uint16 register_value1, register_value2;

    /* Clear CPU DMA interrupt */
    register_value1 = IFR0;
    IFR0 = register_value1;

    /* Read DMA interrupt flags */
    register_value1 = DMA_IFR;

    /* Channels 0-1, input */
    if (register_value1 & 0x0030) // if DMA 1 channel 0 and 1
interrupts are flagged
    {
        register_value2 = DMA1_CH0_TCR2;
        if (register_value2 & 0x0002) {
            PingPongFlagInL = 1; // Last Transfer complete was Pong -
Filling Ping
        } else {
            PingPongFlagInL = 0; // Last Transfer complete was Ping -
Filling Pong
        }

        register_value2 = DMA1_CH1_TCR2;
        if (register_value2 & 0x0002) {
            PingPongFlagInR = 1; // Last Transfer complete was Pong -
Filling Ping
        } else {

```

```

        PingPongFlagInR = 0; // Last Transfer complete was Ping -
        Filling Pong
    }

    /* Clear CH0-1 interrupts */
    DMA_IFR = 0x0030;
}

/* Channels 2-3, output */
if (register_value1 & 0x00C0) // if DMA 1 channel 2-3 interrupts
are flagged
{
    register_value2 = DMA1_CH2_TCR2;
    if (register_value2 & 0x0002) {
        PingPongFlagOutL = 1;
    } else {
        PingPongFlagOutL = 0;
    }

    register_value2 = DMA1_CH3_TCR2;
    if (register_value2 & 0x0002) {
        PingPongFlagOutR = 1;
    } else {
        PingPongFlagOutR = 0;
    }

    /* Clear CH2-3 interrupts */
    DMA_IFR = 0x00C0;
}

}

/* Change word address to byte address and add DARAM offset for DMA */
Uint32 convert_address(Int16 *buffer) {

    Uint32 dma_address;

    dma_address = (Uint32)buffer;
    dma_address = (dma_address<<1) + 0x010000;

    return dma_address;
}

```

My_UART.c

<pre> /* ===== ===== </pre>

```

*   Author: Nathan Zorndorf
*   Description: UART Initialization and Test
*/

#include <stdio.h>
#include <csl_general.h>
#include <csl_uart.h>
#include <My_PLL.h>
#include <usbstk5505.h>
#include <My_UART.h>
#include <Application_1_Modified_Registers.h>

// UART #define's
#define TIMEOUT_VALUE 1000
#define COUNT          3
#define ONE_HALF_SECOND 500000 // 500,000 microseconds in 0.5
seconds

// MIDI #define's
#define NOTE_ON          144 // 0x90 - Note ON for Channel 0 - This
should be the first byte in a Note ON sequence.
#define NOTE_OFF         128 // 0x80 - Note OFF for Channel 0 - This
should be the first byte in a Note OFF sequence.
#define NOTE_VALUE       96 // 0x60 -
#define NOTE_ON_VELOCITY 100 // 0x64 - Velocity = 100 (Range is 1 -
127) - This should be the third byte in a Note ON sequence.
#define NOTE_OFF_VELOCITY 127 // 0x7F - Velocity = 127 (Range is 1 -
127) - This should be the third byte in a Note OFF sequence.

CSL_UartObj      uartObj; // Used to create the UART instance handle.
CSL_UartHandle   hUart;   // The UART instance handle.

int My_UART(void){

    // UART Variable Declaration
    int j = 0;
    Int16      status; // For checking if functions ran
successfully.
    CSL_UartConfig Config;
    char pBuf[3]; // UART TX Buffer - char is 1 byte - ?

    CSL_UartSetup uartSetup =
    {
        // Input clock freq in MHz
        100000000,
        // Baud rate
        31250,
        // Word length of 8
        CSL_UART_WORD8,
        // To generate 1 stop bit

```

```

    0,
    // Disable the parity
    CSL_UART_DISABLE_PARITY,
    // Enable trigger 14 fifo
    CSL_UART_FIFO_DISABLE, // - ??
    // Loop Back enable
    CSL_UART_NO_LOOPBACK,
    // No auto flow control
    CSL_UART_NO_AFE , // - does not matter, choose OFF anyway.
    // No RTS
    CSL_UART_NO_RTS , // - does not matter because AFE is OFF,
choose NO_RST anyway.
};

Config.DLL = 200; // Set baud rate
Config.DLH = 0;
Config.FCR = 0x0000; // Clear UART TX & RX FIFOs
Config.LCR = 0x0003; // 8-bit words, 1 STOP bit
generated, No Parity, No Stick parity, No Break control
Config.MCR = 0x0000; // RTS & CTS disabled, Loopback
mode disabled, Autoflow disabled

// Initialize UART
status = UART_init(&uartObj, CSL_UART_INST_0, UART_POLLED);
if(CSL_SOK != status) { printf("UART_init failed error code
%d\n",status); return(status); }
else { printf("UART_init Successful\n"); }

hUart = (CSL_UartHandle)&uartObj;

// UART Setup
status = UART_setup(hUart, &uartSetup);
if(CSL_SOK != status) { printf("UART_setup failed error code
%d\n",status); return(status); }
else { printf("UART_setup Successful\n"); }

// External Bus Selection Register
CPU_EBSR = 0x1000; // Mode 1 (SPI, GPIO, UART, and I2S2) - OK

// Repeatedly output MIDI ON and MIDI OFF

while(j < 2)
{
    j++;
    pBuf[0] = 0x90;
    pBuf[1] = 0x60;
}

```



```

    pBuf[2]    = 0x64;
    status = UART_fputc(hUart, pBuf[0], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[1], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[2], TIMEOUT_VALUE);
    printf("Sent note ON message viat UART.\n\n");
    USBSTK5505_waitusec(ONE_HALF_SECOND);

    pBuf[0]    = 0x80;
    pBuf[1]    = 0x60;
    pBuf[2]    = 0x7F;
    status = UART_fputc(hUart, pBuf[0], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[1], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[2], TIMEOUT_VALUE);
    printf("Sent note OFF message viat UART.\n\n");
    USBSTK5505_waitusec(ONE_HALF_SECOND);

}

printf("My_UART.c complete.\n");

return(0);

}

```

My_I2S_Register.c

```

/*
=====
=====
*   Author: Nathan Zorndorf
*   Description: Setting up the I2S and AIC3204
*   I2S Transmit and receive interrupts/events are continuously
generated after every transfer from the transmit
data registers to the transmit buffer register and from the receive
buffer register to the receive data
registers respectively. If packed mode is enabled (PACK = 1 in
I2SSCTRL), interrupts/events are
generated after all the required number of data words have been
transmitted/received (see
Section 10.2.8).
From the tech ref (page 351): "Each I2S bus has access to one of the
four DMA peripherals on the DSP"
DMA0 <====> I2S0
DMA1 <====> I2S2
DMA2 <====> I2S3
DMA3 <====> I2S1
*/

```

```

#include <stdio.h>
#include <Application_1_Modified_Registers.h>
#include <tistdtypes.h>

int My_I2S_Register(void) {
    Uint16 register_value;

    // ----- I2S2 Setup Begin -----
    //
    printf("\nI2S SETUP BEGIN!!!\n");

    I2S2_I2SINTMASK = 0x0003; // OUERR (overrun/underrun) and FERR
    (frame sync) error enabled.

    I2S2_I2SSCTRL = 0x0090; // Set I2S0 in stereo mode, 16-bit, with
    data packing, slave

    // Enable I2S0
    register_value = I2S2_I2SSCTRL;
    I2S2_I2SSCTRL = 0x8000 | register_value; // Enable I2S2

    printf("I2S2_I2SINTMASK = 0x%X\n", I2S2_I2SINTMASK);
    printf("I2S2_I2SINTFL = 0x%X\n", I2S2_I2SINTFL);
    printf("I2S2_SRGR = 0x%X \n", I2S2_SRGR); // Print value
of I2S2_SRGR : I2S2 Sample Rate Generator Register
    printf("I2S2_I2SSCTRL = 0x%X \n", I2S2_I2SSCTRL); // Print
value of I2S2_I2SSCTRL : I2S2 Serializer Control Register
    printf("I2S SETUP END!!! \n\n");
    // ----- I2S2 Setup End -----
    //

    return(0);
}

```

Audio_To_MIDI_Using_DMA_and_CFFT.c

```

/* =====
 * Author: Nathan Zorndorf
 * Description: Uses DMA Controller 1 Channels 0-3 to take audio in
from I2S2 RX, do processing and return the fundamental frequency of
the input audio signal.
 */

#include <stdio.h>
#include <math.h>
#include <tms320.h>

```

```

#include <Dsplib.h>
#include <usbstk5505.h>
#include <Application_1_Modified_Registers.h>
#include <Audio_To_MIDI_Using_DMA_and_CFFT.h>
#include <hwafft.h>
#include <Output_MIDI.h>

Int16 OverlapInL[WND_LEN];
Int16 OverlapInR[WND_LEN];
Int16 OverlapOutL[OVERLAP_LENGTH];
Int16 OverlapOutR[OVERLAP_LENGTH];

/* --- buffers required for processing ----*/
#pragma DATA_SECTION(BufferL,"BufL");
Int16 BufferL[FFT_LENGTH];
#pragma DATA_SECTION(BufferR,"BufR");
Int16 BufferR[FFT_LENGTH];
#pragma DATA_SECTION(realL, "rfftL");
Int16 realL[FFT_LENGTH];
#pragma DATA_SECTION(realR, "rfftR");
Int16 realR[FFT_LENGTH];
#pragma DATA_SECTION(imagL, "ifftL");
Int16 imagL[FFT_LENGTH];
#pragma DATA_SECTION(imagR, "ifftR");
Int16 imagR[FFT_LENGTH];
#pragma DATA_SECTION(PSD_Result, "PSD");
Int16 PSD_Result[FFT_LENGTH];
#pragma DATA_SECTION(PSD_Result_sqrt, "PSD_sqrt");
Int16 PSD_Result_sqrt[FFT_LENGTH];
/* -----*/
/* --- Special buffers required for CFFT ----*/
#pragma DATA_SECTION(complex_data,"cmplxBuf");
LDATA complex_data[2*FFT_LENGTH];
/* -----*/

int Audio_To_MIDI_Using_DMA_and_CFFT(void) {

    int i = 0;
    int j = 0;
    int f = 0;
    DATA Peak_Magnitude_Value = 0;
    DATA Peak_Magnitude_Index = 0;
    int MIDI[256] = {0};

    /* Initialize buffers */
    for (i = 0; i < WND_LEN; i++) {
        OverlapInL[i] = 0;
        OverlapInR[i] = 0;
    }
    for (i = 0; i < OVERLAP_LENGTH; i++) {

```

```

    OverlapOutL[i] = 0;
    OverlapOutR[i] = 0;
}

/* Begin infinite loop */
while (1)
{
    /* Get new input audio block */
    if (PingPongFlagInL && PingPongFlagInR) // Last Transfer
complete was Pong – Filling Ping
    {
        for (i = 0; i < HOP_SIZE; i++) {
            /* Copy previous NEW data to current OLD data */
            OverlapInL[i] = OverlapInL[i + HOP_SIZE];
            OverlapInR[i] = OverlapInR[i + HOP_SIZE];

            /* Update NEW data with current audio in */
            OverlapInL[i + HOP_SIZE] = DMA_InpL[i +
AUDIO_IO_SIZE]; // CPU Copies Second Half of index values ("Pong"),
while DMA fills First Half ("Ping")
            OverlapInR[i + HOP_SIZE] = DMA_InpR[i +
AUDIO_IO_SIZE];
        }
    }
    else // Last
Transfer complete was Ping – Filling Pong
    {
        for (i = 0; i < HOP_SIZE; i++) {
            /* Copy previous NEW data to current OLD data */
            OverlapInL[i] = OverlapInL[i + HOP_SIZE];
            OverlapInR[i] = OverlapInR[i + HOP_SIZE];

            /* Update NEW data with current audio in */
            OverlapInL[i + HOP_SIZE] = DMA_InpL[i];
            OverlapInR[i + HOP_SIZE] = DMA_InpR[i];
        }
    }

    /* Create windowed/not windowed buffer for processing */
    for (i = 0; i < WND_LEN; i++) {
        BufferL[i] = OverlapInL[i];
        BufferR[i] = OverlapInR[i];
    }

    /* Convert real data to "pseudo"-complex data (real, 0) */
    /* Int32 complex = Int16 real (MSBs) + Int16 imag (LSBs) */
    for (i = 0; i < FFT_LENGTH; i++)
    {

```

```

        complex_data[2*i] = BufferR[i]; // place audio data (Real)
in each even index of complex_data
        complex_data[2*i+1] = 0;          // place a 0 (Imag) in each
odd index of complex_data
    }

    /* Perform FFT */
    //cfft32(complex_data, HOP_SIZE, SCALE);
    cfft32_SCALE(complex_data, FFT_LENGTH);

    /* Perform bit-reversing */
    cbrev32(complex_data, complex_data, FFT_LENGTH);

    /* Extract real and imaginary parts */
    for (i = 0; i < FFT_LENGTH; i++) {
        realR[i] = complex_data[2*i];
        imagR[i] = complex_data[2*i+1];
    }

    // Find the Power of the audio signal using the cfft results and
scale by 1/2
    for(i = 0; i < FFT_LENGTH; i++) { // square the real vector and
the imaginary vector
        realR[i] = realR[i] * realR[i];
        imagR[i] = imagR[i] * imagR[i];
    }

    // ADD
    for(i = 0; i < FFT_LENGTH; i++) {
        PSD_Result[i] = realR[i] + imagR[i];
    }

    // Scale result by dividing again, because im not sure if I have
the sqrt C library runtime fuccion
    for(i = 0; i < FFT_LENGTH; i++) {
        PSD_Result_sqrt[i] = PSD_Result[i];
    }

    // Process freq. bins from 0Hz to Nyquist frequency (for
efficiency)
    Peak_Magnitude_Value = PSD_Result_sqrt[1]; // start the search at
the first value in the Magnitude spectrum
    Peak_Magnitude_Index = 1;
    for( j = 2; j < NUM_BINS; j++ ) // go through useful frequency

```

```

bins (FFT_LENGTH/2 +1) because the rest are symmetric
{
    if( PSD_Result_sqrt[j] > Peak_Magnitude_Value ) // Peak
search on the magnitude of the FFT to find the fundamental frequency -
the frequency bin with the highest power value in it
    {
        Peak_Magnitude_Value = PSD_Result[j];
        Peak_Magnitude_Index = j;
    }
}

// This huge if-else statement only applies for a sample rate of
8000 samples/sec, and an FFT length of 512.
f++;

if ((Peak_Magnitude_Index == 15)) {
    MIDI[f] = 59; }
else if ((Peak_Magnitude_Index == 16)) {
    MIDI[f] = 60; }
else if ((Peak_Magnitude_Index == 17)) {
    MIDI[f] = 61; }
else if ((Peak_Magnitude_Index == 18)) {
    MIDI[f] = 62; }
else if ((Peak_Magnitude_Index >= 19) && (Peak_Magnitude_Index
<= 20)) {
    MIDI[f] = 63; }
else if ((Peak_Magnitude_Index == 21)) {
    MIDI[f] = 64; }
else if ((Peak_Magnitude_Index == 22)) {
    MIDI[f] = 65; }
else if ((Peak_Magnitude_Index == 23)) {
    MIDI[f] = 66; }
else if ((Peak_Magnitude_Index == 24) || (Peak_Magnitude_Index
== 25)) {
    MIDI[f] = 67; }
else if ((Peak_Magnitude_Index == 26)) {
    MIDI[f] = 68; }
else if ((Peak_Magnitude_Index == 27) || (Peak_Magnitude_Index
== 28)) {
    MIDI[f] = 69; }
else if ((Peak_Magnitude_Index == 29) || (Peak_Magnitude_Index
== 30)) {
    MIDI[f] = 70; }
else if ((Peak_Magnitude_Index >= 31)) {
    MIDI[f] = 71; }
else if ((Peak_Magnitude_Index >= 32) || (Peak_Magnitude_Index
< 34)) {
    MIDI[f] = 72; }
else if ((Peak_Magnitude_Index >= 34) && (Peak_Magnitude_Index

```

```

< 36)) {
    MIDI[f] = 73; }
    else if ((Peak_Magnitude_Index >= 36) && (Peak_Magnitude_Index
< 38)) {
        MIDI[f] = 74; }

    if ((Peak_Magnitude_Index >= 38) && (Peak_Magnitude_Index <
41)) {
        MIDI[f] = 75; }
    else if ((Peak_Magnitude_Index >= 41) && (Peak_Magnitude_Index
< 43)) {
        MIDI[f] = 76; }
    else if ((Peak_Magnitude_Index >= 43) && (Peak_Magnitude_Index
< 46)) {
        MIDI[f] = 77; }
    else if ((Peak_Magnitude_Index >= 46) && (Peak_Magnitude_Index
< 48)) {
        MIDI[f] = 78; }
    else if ((Peak_Magnitude_Index >= 48) && (Peak_Magnitude_Index
< 51)) {
        MIDI[f] = 79; }
    else if ((Peak_Magnitude_Index >= 51) && (Peak_Magnitude_Index
< 55)) {
        MIDI[f] = 80; }
    else if ((Peak_Magnitude_Index >= 55) && (Peak_Magnitude_Index
< 58)) {
        MIDI[f] = 81; }
    else if ((Peak_Magnitude_Index >= 58) && (Peak_Magnitude_Index
< 61)) {
        MIDI[f] = 82; }
    else if ((Peak_Magnitude_Index >= 61) && (Peak_Magnitude_Index
< 65)) {
        MIDI[f] = 83; }
    else if ((Peak_Magnitude_Index >= 65) && (Peak_Magnitude_Index
< 69)) {
        MIDI[f] = 84; }
    else if ((Peak_Magnitude_Index >= 69) && (Peak_Magnitude_Index
< 73)) {
        MIDI[f] = 85; }

    else if ((Peak_Magnitude_Index >= 73) && (Peak_Magnitude_Index
< 77)) {
        MIDI[f] = 86; }
    else if ((Peak_Magnitude_Index >= 77) && (Peak_Magnitude_Index
< 82)) {
        MIDI[f] = 87; }
    else if ((Peak_Magnitude_Index >= 82) && (Peak_Magnitude_Index
< 86)) {
        MIDI[f] = 88; }

```

```

    else if ((Peak_Magnitude_Index >= 86) && (Peak_Magnitude_Index
< 92)) {
        MIDI[f] = 89; }
    else if ((Peak_Magnitude_Index >= 92) && (Peak_Magnitude_Index
< 98)) {
        MIDI[f] = 90; }
    else if ((Peak_Magnitude_Index >= 98) && (Peak_Magnitude_Index
< 104)) {
        MIDI[f] = 91; }
    else {
        MIDI[f] = 0; }

    if(Peak_Magnitude_Value <= POWER_THRESHOLD)
    {
        MIDI[f] = 0;
    }

    Output_MIDI(MIDI[f]); // output the MIDI note through UART.

    if(f == 512) {
        f = 0;
    }

    if (PingPongFlagOutL && PingPongFlagOutR) // Last Transfer
complete was Pong - Filling Ping
    {
        for (i = 0; i < OVERLAP_LENGTH; i++)
        {
            /* Current output block is previous overlapped block +
current processed block */
            DMA_OutL[i + AUDIO_IO_SIZE] = OverlapOutL[i] +
BufferL[i];
            DMA_OutR[i + AUDIO_IO_SIZE] = OverlapOutR[i] +
BufferR[i];

            /* Update overlap buffer */
            OverlapOutL[i] = BufferL[i + HOP_SIZE];
            OverlapOutR[i] = BufferR[i + HOP_SIZE];
        }
    }
    else // Last
Transfer complete was Ping - Filling Pong
    {
        for (i = 0; i < OVERLAP_LENGTH; i++)
        {
            /* Current output block is previous overlapped block +
current processed block */
            DMA_OutL[i] = OverlapOutL[i] + BufferL[i];
            DMA_OutR[i] = OverlapOutR[i] + BufferR[i];

```



```

        /* Update overlap buffer */
        OverlapOutL[i] = BufferL[i + HOP_SIZE];
        OverlapOutR[i] = BufferR[i + HOP_SIZE];
    }
}
}
}
}

```

Output_MIDI.c

```

/*
=====
=====
*   Author: Nathan Zorndorf
*   Description: UART Initialization and Test
*/

#include <stdio.h>
#include <csl_general.h>
#include <csl_uart.h>
#include <usbstk5505.h>

// UART #define's
#define TIMEOUT_VALUE 1000
#define COUNT          3
#define ONE_HALF_SECOND 500000 // 500,000 microseconds in 0.5
seconds

// MIDI #define's
#define NOTE_ON          144 // 0x90 - Note ON for Channel 0 - This
should be the first byte in a Note ON sequence.
#define NOTE_OFF          128 // 0x80 - Note OFF for Channel 0 - This
should be the first byte in a Note OFF sequence.
#define NOTE_VALUE        96 // 0x60 -
#define NOTE_ON_VELOCITY  100 // 0x64 - Velocity = 100 (Range is 1 -
127) - This should be the third byte in a Note ON sequence.
#define NOTE_OFF_VELOCITY 127 // 0x7F - Velocity = 127 (Range is 1 -
127) - This should be the third byte in a Note OFF sequence.

extern CSL_UartHandle hUart;
int Current_MIDI_Number;
int Previous_MIDI_Number;

int Output_MIDI(int MIDI_Number){
    Int16 status;

```

```

int i;
char pBuf[3]; // UART TX Buffer - char is 1 byte - ?

pBuf[2] = NOTE_ON_VELOCITY; // hardcode the velocity value of each
note to be 100

Current_MIDI_Number = MIDI_Number; // make the input MIDI_Number
equal to the current MIDI number

if ((Current_MIDI_Number > 91) || (Current_MIDI_Number < 59)) {
    Current_MIDI_Number == Previous_MIDI_Number;
}

if((Current_MIDI_Number != Previous_MIDI_Number) &&
(Current_MIDI_Number != 0)) {
    pBuf[0] = NOTE_OFF; // turn OFF the PREVIOUS MIDI
note
    pBuf[1] = Previous_MIDI_Number;
    pBuf[2] = 100; // Note OFF velocity is 0
    status = UART_fputc(hUart, pBuf[0], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[1], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[2], TIMEOUT_VALUE);

    pBuf[0] = NOTE_ON; // turn ON the CURRENT MIDI
note
    pBuf[1] = Current_MIDI_Number;
    pBuf[2] = NOTE_ON_VELOCITY;
    status = UART_fputc(hUart, pBuf[0], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[1], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[2], TIMEOUT_VALUE);
}

else if ((Previous_MIDI_Number != 0) && (Current_MIDI_Number == 0))
{
    pBuf[0] = NOTE_OFF; // turn OFF the PREVIOUS MIDI
note
    pBuf[1] = Previous_MIDI_Number;
    pBuf[2] = 100; // Note OFF velocity is 0
    status = UART_fputc(hUart, pBuf[0], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[1], TIMEOUT_VALUE);
    status = UART_fputc(hUart, pBuf[2], TIMEOUT_VALUE);
}

else if ((Previous_MIDI_Number == Current_MIDI_Number)) { // dummy
- basically do nothing if this is true
    i++;
    if (i == 500) {
        i = 0;
    }
}

```

```

    //USBSTK5505_waitusec(100000); // smooth the erratic output MIDI
messages

    Previous_MIDI_Number = Current_MIDI_Number;

    return Current_MIDI_Number;
}

```

Aic3204.c – written by [Richard Sikora](#)

```

/*****
*****/
/*
*/
/* FILENAME
*/
/*  aic3204.c
*/
/*
*/
/* DESCRIPTION
*/
/*  Setup functions for aic3204 codec on the TMS320C5505 USB Stick.
*/
/*
*/
/* REVISION
*/
/*  Revision: 1.00
*/
/*  Author   : Richard Sikora
*/
/*-----
-----*/
/*
*/
/* HISTORY
*/
/*
*/
/*  Revision 1.00
*/
/*  20th December 2009. Created by Richard Sikora from Spectrum
Digital      */
/*                      code. Created new functions for codec read
*/
/*                      and write.

```

```

*/
/*
*/
/*****
*****/
/*
 *
 * Copyright (C) 2010 Texas Instruments Incorporated -
http://www.ti.com/
 *
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 *   Redistributions in binary form must reproduce the above
copyright
 *   notice, this list of conditions and the following disclaimer in
the
 *   documentation and/or other materials provided with the
 *   distribution.
 *
 *   Neither the name of Texas Instruments Incorporated nor the names
of
 *   its contributors may be used to endorse or promote products
derived
 *   from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH

```

```

DAMAGE.
*
*/

#include "usbstk5505.h"
#include "aic3204.h"
#include "usbstk5505_gpio.h"
#include "usbstk5505_i2c.h"

Int16 counter1; // Counters for monitoring real-time operation.
Int16 counter2;

/* -----
*
*
* _AIC3204_rget( regnum, regval )
*
*
*
*      Return value of codec register regnum
*
*
* -----
*/
Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
    Int16 retcode = 0;
    Uint8 cmd[2];

    cmd[0] = regnum & 0x007F;      // 7-bit Device Address
    cmd[1] = 0;

    retcode |= USBSTK5505_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
    retcode |= USBSTK5505_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );

    *regval = cmd[0];
    USBSTK5505_wait( 10 );
    return retcode;
}

/* -----
*
*
* _AIC3204_rset( regnum, regval )
*
*
*

```

```

*
*      Set codec register regnum to value regval
*
*
*
* ----- */
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
    Uint8 cmd[2];
    cmd[0] = regnum & 0x007F;      // 7-bit Device Address
    cmd[1] = regval;               // 8-bit Register Data

    return USBSTK5505_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}

/* ----- */
*
*
*      aic3204_enable( )
*
*
*
* ----- */
void aic3204_hardware_init(void)
{
    // SYS_EXBUSSEL |= 0x0020; // Select A20/GPIO26 as GPIO26
    // USBSTK5505_GPIO_init();
    // USBSTK5505_GPIO_setDirection(GPIO26, GPIO_OUT);
    // USBSTK5505_GPIO_setOutput( GPIO26, 1 ); // Take AIC3204 chip out
of reset
    USBSTK5505_I2C_init( ); // Initialize I2C
    USBSTK5505_wait( 100 ); // Wait
}

/* ----- */
*
*
*      aic3204_disable( )
*
*
*
* ----- */

```

```

void aic3204_disable(void)
{
    AIC3204_rset( 1, 1 );           // Reset codec
    // USBSTK5505_GPIO_setOutput( GPIO26, 0 ); // Put AIC3204 into reset
    I2S2_CR = 0x00;
}

/* -----
 *
 *
 * aic3204_codec_read( )
 *
 *
 * ----- */

void aic3204_codec_read(Int16* left_input, Int16* right_input)
{
    volatile Int16 dummy;

    counter1 = 0;

    /* Read Digital audio inputs */
    while(!(I2S2_IR & RcvR) ) // != NO Stereo receive interrupt
    pending. DO NOT Read Receive Left and Right data 0 and 1 registers.
    {
        counter1++; // Wait for receive interrupt
    }

    *left_input = I2S2_W0_MSW_R;           // Read Most Significant Word
    of first channel
    dummy = I2S2_W0_LSW_R;                 // Read Least Significant
    Word (ignore)
    *right_input = I2S2_W1_MSW_R;          // Read Most Significant Word
    of second channel
    dummy = I2S2_W1_LSW_R;                 // Read Least Significant
    Word of second channel (ignore)
}

/* -----
 *
 *
 * aic3204_codec_write( )
 *
 *
 *

```

```

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

void aic3204_codec_write(Int16 left_output, Int16 right_output)
{
    counter2 = 0;

    while( !(I2S2_IR & XmitR) )
    {
        counter2++; // Wait for transmit interrupt
    }
    I2S2_W0_MSW_W = left_output;           // Left output
    I2S2_W0_LSW_W = 0;
    I2S2_W1_MSW_W = right_output;         // Right output
    I2S2_W1_LSW_W = 0;
}

/* ----- */
----- *
*
*
*   End of aic3204.c
*
*
*
* ----- */
----- */

```

Usbstk5505.c

```

/*****
*****/
/*
*/
/* FILENAME
*/
/*   aic3204.c
*/
/*
*/
/* DESCRIPTION
*/
/*   Setup functions for aic3204 codec on the TMS320C5505 USB Stick.
*/
/*
*/
/* REVISION
*/
/*   Revision: 1.00

```



```

*/
/*  Author   : Richard Sikora
*/
/*-----
-----*/
/*
*/
/* HISTORY
*/
/*
*/
/* Revision 1.00
*/
/* 20th December 2009. Created by Richard Sikora from Spectrum
Digital      */
/*                      code. Created new functions for codec read
*/
/*                      and write.
*/
/*
*/
/*
*/
/*****
*****/
/*
*
* Copyright (C) 2010 Texas Instruments Incorporated -
http://www.ti.com/
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
*
*   Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
*
*   Redistributions in binary form must reproduce the above
copyright
*   notice, this list of conditions and the following disclaimer in
the
*   documentation and/or other materials provided with the
*   distribution.
*
*   Neither the name of Texas Instruments Incorporated nor the names
of
*   its contributors may be used to endorse or promote products
derived
*   from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND

```

CONTRIBUTORS

```

* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
*
*/

```

```

#include "usbstk5505.h"
#include "aic3204.h"
#include "usbstk5505_gpio.h"
#include "usbstk5505_i2c.h"

```

```

Int16 counter1; // Counters for monitoring real-time operation.
Int16 counter2;

```

```

/* -----
----- *
*
*
* _AIC3204_rget( regnum, regval )
*
*
*
*      Return value of codec register regnum
*
*
*
* -----
----- */
Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
    Int16 retcode = 0;
    Uint8 cmd[2];

```

```

    cmd[0] = regnum & 0x007F;          // 7-bit Device Address
    cmd[1] = 0;

    retcode |= USBSTK5505_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
    retcode |= USBSTK5505_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );

    *regval = cmd[0];
    USBSTK5505_wait( 10 );
    return retcode;
}

/* -----
 *
 *
 * _AIC3204_rset( regnum, regval )
 *
 *
 *      Set codec register regnum to value regval
 *
 *
 * -----
 */
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
    Uint8 cmd[2];
    cmd[0] = regnum & 0x007F;          // 7-bit Device Address
    cmd[1] = regval;                   // 8-bit Register Data

    return USBSTK5505_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}

/* -----
 *
 *
 * aic3204_enable( )
 *
 *
 * -----
 */

void aic3204_hardware_init(void)
{
    // SYS_EXBUSSEL |= 0x0020; // Select A20/GPI026 as GPI026
    // USBSTK5505_GPIO_init();

```

```

// USBSTK5505_GPIO_setDirection(GPIO26, GPIO_OUT);
// USBSTK5505_GPIO_setOutput( GPIO26, 1 );    // Take AIC3204 chip out
of reset
USBSTK5505_I2C_init( );                        // Initialize I2C
USBSTK5505_wait( 100 );    // Wait
}

/* -----
 *
 *
 *   aic3204_disable( )
 *
 *
 * ----- */

void aic3204_disable(void)
{
    AIC3204_rset( 1, 1 );                        // Reset codec
//   USBSTK5505_GPIO_setOutput( GPIO26, 0 ); // Put AIC3204 into reset
    I2S2_CR = 0x00;
}

/* -----
 *
 *
 *   aic3204_codec_read( )
 *
 *
 * ----- */

void aic3204_codec_read(Int16* left_input, Int16* right_input)
{
    volatile Int16 dummy;

    counter1 = 0;

    /* Read Digital audio inputs */
    while(!(I2S2_IR & RcvR) ) // != NO Stereo receive interrupt
pending. DO NOT Read Receive Left and Right data 0 and 1 registers.
    {
        counter1++; // Wait for receive interrupt
    }

    *left_input = I2S2_W0_MSW_R;                // Read Most Significant Word

```

```

of first channel
    dummy = I2S2_W0_LSW_R;           // Read Least Significant
Word (ignore)
    *right_input = I2S2_W1_MSW_R;     // Read Most Significant Word
of second channel
    dummy = I2S2_W1_LSW_R;           // Read Least Significant
Word of second channel (ignore)

}

/* -----
----- *
*
*
*   aic3204_codec_write( )
*
*
* -----
----- */

void aic3204_codec_write(Int16 left_output, Int16 right_output)
{
    counter2 = 0;

    while( !(I2S2_IR & XmitR) )
    {
        counter2++; // Wait for transmit interrupt
    }
    I2S2_W0_MSW_W = left_output;      // Left output
    I2S2_W0_LSW_W = 0;
    I2S2_W1_MSW_W = right_output;     // Right output
    I2S2_W1_LSW_W = 0;
}

/* -----
----- *
*
*
*   End of aic3204.c
*
*
* -----
----- */

```

Cfft32_scale.asm – the FFT assembly language (optimized) source file - From Texas Instruments

```

; /*
; * Copyright (C) 2013 Texas Instruments Incorporated - http://www.ti.com/
; *
; *
; * Redistribution and use in source and binary forms, with or without
; * modification, are permitted provided that the following conditions
; * are met:
; *
; *
; * Redistributions of source code must retain the above copyright
; * notice, this list of conditions and the following disclaimer.
; *
; *
; * Redistributions in binary form must reproduce the above copyright
; * notice, this list of conditions and the following disclaimer in the
; * documentation and/or other materials provided with the
; * distribution.
; *
; *
; * Neither the name of Texas Instruments Incorporated nor the names of
; * its contributors may be used to endorse or promote products derived
; * from this software without specific prior written permission.
; *
; * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
; * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
; * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
; * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
; * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
; * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
; * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
; * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
; * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
; * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
; * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
; *
; */

; *****
; Version 3.00.00
; *****
; Processor: C55xx
; Description: 32-bit radix-2 DIT complex FFT using normal input data
; and bit-reversed twiddle table (length N/2, cosine/sine format)
; First two stages are separately implmented for MIPS optimization.
;
; Usage: void cfft32_SCALE (LDATA *x, ushort nx);
;
; Limitations:
; nx must be a power of 2 between 8 and 1024
;

```

```

; Benchmarks:
; Cycles:
;   nx = 8: 250
;   nx = 16: 575
;   nx = 32: 1349
;   nx = 64: 3111
;   nx =128: 7159
;   nx =256: 16236
;   nx =512: 36396
; Code Size (in bytes):
;   .text      496
;   .const:twiddle32 4096 (shared by cfft32, cfft32, unpack32, unpacki32)
;
; History:
;   Original: 08/16/2002 ZhengTing He
;   08/19/2002 Li Yuan
;     - Changed || to :: in several dual MAC instructions
;   06/19/2012 Craig Leeds
;     - optimized
;*****
;-----
; Arguments passed to _fft
;
; AR0    ->  fftdata pointer
; T0     ->  fft size
;
;-----

;-----
; Array declarations
;-----

.ref twiddle32
.def _cfft32_SCALE          ; make function visible to other fnct
.cpl_on
.arms_off                  ; enable assembler for arms mode
.mmregs
.noremark 5579, 5573, 5684

; Stack frame
; -----
RET_ADDR_SZ    .set 1      ;return address
REG_SAVE_SZ    .set 5      ;save-on-entry registers saved (T3, T2, AR5, AR6, AR7)
FRAME_SZ       .set 2      ;local variables
ARG_BLK_SZ     .set 0      ;argument block

```

```
PARAM_OFFSET    .set ARG_BLK_SZ + FRAME_SZ + REG_SAVE_SZ + RET_ADDR_SZ
```

```
; Local variables
```

```
;-----
```

```
    .asg  0, data_ptr
```

```
    .asg  1, data_sz
```

```
;-----
```

```
; Conditional compile
```

```
;-----
```

```
SCALED    .set    1            ; SCALED = 0 - not scaled version
                                ; SCALED = 1 - scaled version
```

```
    .text
```

```
_cfft32_SCALE:
```

```
;-----
```

```
; Save any save-on-entry registers that are used
```

```
;-----
```

```
    PSH   T3, T2
```

```
    || BSET  #FRCT, ST1_55
```

```
    PSHBOTH XAR5
```

```
    || BCLR  #ARMS, ST2_55
```

```
    PSHBOTH XAR6
```

```
    || BSET  M40
```

```
    PSHBOTH XAR7
```

```
;-----
```

```
; Allocate the local frame and argument block
```

```
;-----
```

```
    AADD   #-(ARG_BLK_SZ + FRAME_SZ), SP
```

```
;-----
```

```
; Save entry values for later
```

```
;-----
```

```
    MOV    AR0, *SP(data_ptr);
```

```
    MOV    T0, *SP(data_sz) ;
```

```
;-----
```

```
; FFT implementation
```

```
;
```

```
; The FFT is implemented in 5 different steps:
```

```
;
```

```
; 1) - a radix-2 stage without any multiplications.
```



```

; 2) - a radix-2 stage with two groups, only the 2nd group has
;   multiplications with 0x7FFFFFFF and 0x00000000
; 3) - a group of log2(FFT_SIZE)-3 radix-2 stages
; 4) - a radix-2 stage without scaling.
; 5) - on out-of-place bit-reversal
;-----

;-----
; Modification of status registers
;-----

        ; Define CSR for scaling loop
SUB     #1, T0, T1
MOV     T1, BRC0          ; BRC0 = fftsize - 1
AMAR    *AR0,XAR1
AMAR    *AR0,XAR2

;-----

; Scaling loop: Data scaled by 2 before first stage
;-----
MOV     dbl(*AR2+), AC0          ; (Prime the Pump)
|| RPTBLOCAL scaling-1
    MOV     dbl(*AR2+), AC1
    || SFTS  AC0,#-1
    SFTS    AC1,#-1
    || MOV   AC0, dbl(*AR1+)
    MOV     AC1, dbl(*AR1+)
    || MOV   dbl(*AR2+), AC0          ; (for next iteration)
scaling:

;-----

; Radix-2 stage 1
;-----

        ; AR0->fft_data (a)
AMAR    *AR0(T0),XAR1          ; AR1->fft_data+1*n2 (b)
AMAR    *AR1(T0),XAR2          ; AR2->fft_data+2*n2 (c)
AMAR    *AR2(T0),XAR3          ; AR3->fft_data+3*n2 (d)
MOV     XAR2,XAR7

SFTS    T0, #-1                ; T0=fft-size/2
SUB     #1, T0, T1              ; T1=fft_size/2-1
MOV     T1, BRC0
AMAR    *AR0, XAR4

;-----

; Benchmark: 9 cycles for this loop
;-----

MOV     dbl(*AR4+),AC0          ; (Prime the Pump)

```

```

|| RPTBLOCAL stage1-1
    ADD  dbl(*AR7),AC0,AC1      ; AC1=AR+CR
    SUB  dbl(*AR7+),AC0        ; AC0=AR-CR
    MOV  dbl(*AR4+),AC2
    || SFTS AC1,#-1
    ADD  dbl(*AR7),AC2,AC3      ; AC3=AI+CI
    SUB  dbl(*AR7-),AC2        ; AC2=AI-CI
    SFTS AC0,#-1
    || MOV  AC1,dbl(*AR0+)      ;(AR+CR)>>1->AR
    SFTS AC2,#-1
    || MOV  AC0,dbl(*AR7+)      ;(AR-CR)>>1->CR
    SFTS AC3,#-1
    || MOV  AC2,dbl(*AR7+)      ;(AI-CI)>>1->CI
    MOV  AC3,dbl(*AR0+)        ;(AI+CI)>>1->AI
    || MOV  dbl(*AR4+),AC0      ; (for next iteration)
stage1:

```

```

;-----
; Radix-2 stage2
;-----

```

```

    MOV  *SP(data_ptr), AR0
    || SFTS T0, #-1
    SUB  #1, T0, T1            ;T0=fft_size/4
    MOV  T1, BRC0              ;T1=fft_size/4-1
    AMAR  *AR0, XAR4

```

```

;-----
; Benchmark: 10 cycles for group1
;-----

```

```

    MOV  dbl(*AR4+),AC0        ; (Prime the Pump)
    || RPTBLOCAL group1-1
    ADD  dbl(*AR1),AC0,AC1      ; AC1=AR+BR
    SUB  dbl(*AR1+),AC0        ; AC0=AR-BR
    || SFTS AC1,#-1
    MOV  dbl(*AR4+),AC2
    ADD  dbl(*AR1),AC2,AC3      ; AC3=AI+BI
    SUB  dbl(*AR1-),AC2        ; AC2=AI-BI
    || SFTS AC0,#-1
    MOV  AC1,dbl(*AR0+)        ; AR+BR->AR
    || SFTS AC2,#-1
    MOV  AC0,dbl(*AR1+)        ; AR-BR->BR
    || SFTS AC3,#-1
    MOV  AC2,dbl(*AR1+)        ; AI-BI->BI
    MOV  AC3,dbl(*AR0+)        ; AI+BI->AI
    || MOV  dbl(*AR4+),AC0      ; (for next iteration)
group1:

```

```

;-----
; Benchmark: 11 cycles for group2
;-----
MOV    T1,BRC0
AMOV   #twiddle32, XCDP
AMAR   *AR3(#3), XAR4          ; AR4->DI_LOW
MOV    #3, T1
AMAR   *AR3+                  ;          AR3->DR_LOW

|| RPTBLOCAL group2-1
    MPY   uns(*AR3-),*CDP+,AC0      ;DR_LOW * 7FFF      AR3->DR_HI  CDP->FFFF
:: MPY   uns(*AR4-),*CDP+,AC1      ;DI_LOW * 7FFF      AR4->DI_HI
    MAC   *AR3,uns(*CDP-),AC0      ;DR_HI*FFFF+DR_LOW*7FFF, AR3->DR_HI  CDP->7FFF
:: MAC   *AR4,uns(*CDP-),AC1      ;DI_HI*FFFF+DI_LOW*7FFF, AR4->DI_HI
    MAC   *AR3,*CDP,AC0>>#16      ;DR_HI*7FFF+AC0>>16  AR3->DR_HI  CDP->7FFF
:: MAC   *AR4,*CDP,AC1>>#16      ;DI_HI*7FFF+AC1>>16  AR4->DI_HI  CDP->7FFF
    SUB   AC1,dbl(*AR2),AC2        ;AC2=CR-AC1
|| ADD   #5, AR4                  ;          AR4->DI(next)
    ADD   dbl(*AR2+),AC1          ;AC1=CR+AC1      AR2->CI
    SUB   AC0,dbl(*AR2),AC3        ;AC3=CI-AC0
|| SFTS  AC1,#-1
    ADD   dbl(*AR2-),AC0          ;AC0=CI+AC0      AR2->CR
|| SFTS  AC3,#-1
    MOV   AC1,dbl(*AR2+)          ;AC1->CR      AR2->CI
|| SFTS  AC2,#-1
    MOV   AC3,dbl(*AR2+)          ;AC3->CI      AR2->CR(next)
|| SFTS  AC0,#-1
    MOV   AC2,dbl(*AR3+)          ;AC2->DR      AR3->DI
    MOV   AC0,dbl(*AR3+T1))      ;AC0->DI      AR3->DR(next)
group2:

;-----
; End of stage 1 and 2
;-----
SFTS   T0, #-1
BCC    end_benchmark, T0==#0

;-----
; radix-2 stages (stages 3->log2(FFT_SIZE) )
;   register usage
;   AR0->Pr, AR1->Qr, AR3->twiddle
;   AR4=Re distance of butterfly
;   AR6=group count, t1=butterfly count, AR5= stage count
;-----
; main initialization

; modify ST2 to select linear or circular addressing modes

```

```

OR    #0x3 , mmap(ST2_55)          ; circular AR0,AR1
MOV    *SP(data_ptr), AR1          ; AR1 = #fftdata

; circular buffer starting addresses
MOV    AR1, mmap(BSA01)            ; circular buffer start address

; circular buffer sizes
MPYMK  *SP(data_sz),#2,AC0          ; because FRCT==1, it actually x4
MOV    AC0, mmap(BK03)             ; bk03 = (4*FFT_SIZE-4), AR0-AR3

MOV    *SP(data_sz), T2
SFTS   T2, #-1                     ; T2 = FFT_SIZE/2
|| MOV  #4, AR6                     ; AR6 = group
MOV    T2, AR4                     ; AR4 = FFT_SIZE/2(Re distance between p q)

MOV    T0, T1                      ; T1 = FFT_SIZE/8, nbfly

SFTS   T0, #-1                     ; T0=size/16
|| MOV  #-2, T2
MOV    T0, AR5                     ; AR5 is stage count
MOV    #2, T0
AMOV   #twiddle32, XAR3
BCC    last_stage, AR5==#0

;-----
; Beginning of the stage loop
;   stage initialization
;-----
stage: ; stage loop counter updates
      SFTS   AR5, #-1              ; shift right stage count

      MOV    #0, AR0
      || MOV  AR3, CDP
      ADD    #1, AR4, AR1          ; AR1->QR_LOW

      ; butterfly counter update
      SUB    #1, T1, T3
      MOV    T3, BRC1

      ; group counter update
      SUB    #1, AR6, T3
      MOV    T3, BRC0

      RPTBLOCAL group-1
;-----
; Benchmark: 15 cycles for butterfly loop
;-----

```

```
RPTBLOCAL BFly-1 ; (AR1,CDP)
```

```
MPY uns(*AR1), *(CDP+T0), AC0 ; AC0 = yrl*crh (1,0)
:: MPY uns(*AR1(T0)), *(CDP+T0), AC1 ; AC1 = yil*crh (3,0)
```

```
MAC uns(*AR1(T0)), *CDP+, AC0 ; AC0 += yil*cih (3,2)
:: MAS uns(*AR1+), *CDP+, AC1 ; AC1 -= yrl*cih (1,2)
|| SWAP T0, T2 ; T0=-2
```

```
MAC *AR1, uns(*(CDP+T0)), AC0 ; AC0 += yih*cil (2,3)
:: MAS *AR1(T0), uns(*(CDP+T0)), AC1 ; AC1 -= yrh*cil (0,3)
```

```
MAC *AR1(T0), uns(*CDP-), AC0 ; AC0 += yrh*crl (0,1)
:: MAC *(AR1+T0), uns(*CDP-), AC1 ; AC1 += yih*crl (2,1)
|| SWAP T0, T2 ; T0=2
```

```
MAC *AR1, *(CDP+T0), AC0>>#16 ; AC0 += yrh*crh (0,0)
:: MAC *AR1(T0), *(CDP+T0), AC1>>#16 ; AC1 += yih*crh (2,0)
```

```
MAC *AR1(T0), *(CDP+T0), AC0 ; AC0 += yih*cih (2,2)
:: MAS *AR1, *(CDP+T0), AC1 ; AC1 -= yrh*cih (0,2)
```

```
ADD dbl(*AR0), AC0, AC2
SFTS AC2, #-1
|| MAR *CDP-
MOV AC2, dbl(*AR0) ; new xr=AC0+xr (0,4)
|| SUB AC0, dbl(*AR0+), AC3 ; (0,4)
SFTS AC3, #-1
|| MAR *CDP-
MOV AC3, dbl(*AR1+) ; new yr=xr-AC0 (2,4)
|| SUB AC1, dbl(*AR0), AC2
SFTS AC2, #-1
|| MAR *CDP-
MOV AC2, dbl(*AR1+) ; new yi=xi-AC1 (2,4)
|| ADD dbl(*AR0), AC1, AC3 ; (4,4)
SFTS AC3, #-1
|| MAR *CDP-
MOV AC3, dbl(*AR0+) ; new xi=xi+AC1
|| AADD #1, AR1 ; (4,4)
```

BFly:

```
ADD AR4, AR0 ;jump to next group
ADD AR4, AR1
AMAR *+CDP(4) ;CDP+4
```

group:

```
SFTS AR6, #1 ;group<<1
SFTS T1, #-1 ;butterfly>>1
SFTS AR4, #-1 ;P Q distance>>1
|| BCC stage, AR5 != #0
```

```

;-----
; END radix-2 stages (stages 3->log2(FFT_SIZE) )
;-----

;-----
; Last stage
;-----
last_stage:
    ; stage initialization
    MOV    #0, AR0
    MOV    AR3, CDP
    MOV    #5, AR1                ;AR1->QR_LOW

    ; group counter update
    SUB    #1, AR6, T3
    MOV    T3, BRC0
    MOV    #6, T1

|| RPTBLOCAL lgroup-1
    MPY    uns(*AR1), *(CDP+T0), AC0    ; AC0 = yrl*crh (1,0)
    :: MPY    uns(*AR1(T0)), *(CDP+T0), AC1    ; AC1 = yil*crh (3,0)

    MAC    uns(*AR1(T0)), *CDP+, AC0    ; AC0 += yil*cih (3,2)
    :: MAS    uns(*AR1+), *CDP+, AC1    ; AC1 -= yrl*cih (1,2)
    || SWAP    T0, T2                ; T0=-2

    MAC    *AR1, uns(*(CDP+T0)), AC0    ; AC0 += yih*cil (2,3)
    :: MAS    *AR1(T0), uns(*(CDP+T0)), AC1    ; AC1 -= yrh*cil (0,3)

    MAC    *AR1(T0), uns(*CDP-), AC0    ; AC0 += yrh*crl (0,1)
    :: MAC    *(AR1+T0), uns(*CDP-), AC1    ; AC1 += yih*crl (2,1)
    || SWAP    T0, T2                ; T0=2

    MAC    *AR1, *(CDP+T0), AC0>>#16    ; AC0 += yrh*crh (0,0)
    :: MAC    *AR1(T0), *(CDP+T0), AC1>>#16    ; AC1 += yih*crh (2,0)

    MAC    *AR1(T0), *(CDP+T0), AC0    ; AC0 += yih*cih (2,2)
    :: MAS    *AR1, *(CDP+T0), AC1    ; AC1 -= yrh*cih (0,2)

    ADD    dbl(*AR0), AC0, AC2
    MOV    AC2, dbl(*AR0)                ; new xr=AC0+xr (0,4)
    || SUB    AC0, dbl(*AR0+), AC3                ; (0,4)

    MOV    AC3, dbl(*AR1+)                ; new yr=xr-AC0 (2,4)
    || SUB    AC1, dbl(*AR0), AC2

    MOV    AC2, dbl(*AR1+)                ; new yi=xi-AC1 (2,4)
    || ADD    dbl(*AR0), AC1, AC3                ; (4,4)

```

```

        MOV   AC3, dbl(*(AR0+T1))      ; new xi=xi+AC1 (and jump to next group)
    || AADD   #5, AR1                  ;      (4,4)
lgroup:
;-----
; END last stage
;-----

end_benchmark:
;-----
; Allocate the local frame and argument block
;-----
    AADD   #+(ARG_BLK_SZ + FRAME_SZ), SP

;Context restore
    POPBOTH XAR7
    || BCLR  AROLC
    POPBOTH XAR6
    || BCLR  AR1LC
    POPBOTH XAR5
    || BSET  #ARMS, ST2_55
    POP    T3, T2
    || BCLR  #FRCT, ST1_55
    RET
    || BCLR  M40
    .end

```

Ezdsp5535.gel – can be found on Spectrum Digital's website for the C5535 support resources

```

/* -----
----- *
*
*
*   * ezdsp5535.gel
*
*   *   Version 1.00
*
*
*
*   *   This GEL file is designed to be used in conjunction with
*
*   *   CCS 4.2 and the eZDSP5535.
*
*
*
*   *   History
*
*

```

```

*      v0.01      Initial Release
*
*      v1.00      Public Release
*
*
*
* ----- */
/* -----
*
*
*      StartUp( )
*
*      This function is called each time CCS is started.
*
*      Setup Memory Map
*
*      Do not touch the target
*
*
* ----- */
StartUp()
{
    GEL_TextOut( "\nStartUp()\n" );
    c5535_MapInit();
}

/* -----
*
*
*      OnTargetConnect( )
*
*      This function is call automatically when you connect to the
target.      *
*      Under normal circumstances it should be used to fully intialize
the      *
*      connected CPU and optionally the rest of the chip or board.
*
*
*
*      Operations that may be needed before the cpu/chip can be
configured      *
*      GEL_Reset

```



```

*
*      Disable MMU - Important for ARM
*
*      Disable DMA
*
*
*
* Note: OnTargetConnect might not be appropriate if you are trying
to      *
*      debug a running OS. In general if debugging a running OS
you      *
*      only want the memory map defined.
*
* -----
----- */
OnTargetConnect()
{
    GEL_TextOut( "\nOnTargetConnect()\n" );

    c5535_MapInit();
    GEL_Reset();
    SystemCleanup();
    ProgramPLL_100MHz();
    GEL_TextOut("Target Connection Complete.\n");
}

/* -----
----- *
*
*
* OnReset( )
*
*      This function is called by CCS when you do a reset or
GEL_Reset.      *
*      It can be used to configure/reconfigure portions of the chip
that      *
*      are cleared by the reset.
*
*
*
* Note: OnReset is generally used by older targets when
reset/GEL_Reset      *
*      actually reset the chip. On newer multi-core devices the
reset      *
*      is basically a SW reset of the connected CPU.
*
*      If you want to catch an external reset or an IcePick reset
then      *
*      OnResetDetected() can be used.
*
*

```

```

* -----
----- */
OnReset( int nErrorCode )
{
    GEL_TextOut( "\nOnReset()\n" );
}

/* -----
----- *
*
*
*   OnRestart( )
*
*       This function is called by CCS when you do Debug->Restart.
*
*       The assumption is that the target has not hung and program is
still *
*       in memory. If that is not the case then program should be
reloaded *
*       vs restarted.
*
*
*
*       Operations that may be needed:
*
*           Disable MMU
*
*           Flush/Disable Caches
*
*           Clear/Disable Interrupts
*
*           Disable DMA
*
*           For ARM set the CPSR. This effects exec mode IRQ/FIQ and
Thumb. *
*           Disable watchdogs
*
* -----
----- */
OnRestart( int nErrorCode )
{
    GEL_TextOut( "\nOnRestart()\n" );
    SystemCleanup();
}

/* -----
----- *
*
*
*   OnPreFileLoaded( )

```

```

*
*   This function is called automatically when the 'Load Program'
*
*   Menu item is selected. This function should ensure that the
cpu  *
*   is in a good enough state so that program, data and i/o memory
is   *
*   readable and writeable. Every chip is different especially
the  *
*   reset so some things have to be done manually especially on
ARM  *
*   devices.
*
*
*
*   Operations that may be needed:
*
*       GEL_Reset
*
*       Disable MMU - Important for ARM
*
*       Disable DMA
*
*       Reconfigure EMIF if GEL_Reset reset it.
*
*
*   Note: OnRestart if present is called after the file is loaded
so   *
*       there is some reduncancy but this can be minimized.
*
* -----
----- */
OnPreFileLoaded()
{
    GEL_TextOut( "\nOnPreFileLoaded()\n" );
    SystemCleanup();
    ProgramPLL_100MHz();
}

/* -----
----- *
*
*
*   SystemCleanup()
*
*   Clean up DSP state
*
*

```

```

*
* -----
----- */
SystemCleanup()
{
    /* Disable interrupts */
    *(int*)0x0003 = *(int*)0x0003 | 0x0800; // Set INTM
    *(int*)0x0000 = 0;                      // Clear IER0
    *(int*)0x0045 = 0;                      // Clear IER1
    *(int*)0x0004 = *(int*)0x0004 | 0x2000; // Set CACLR (Clear Cache)

    /* Disable each DMA channels */
    *(int*)0xC01@io = 0;    // DMA0
    *(int*)0xC21@io = 0;    // DMA1
    *(int*)0xC41@io = 0;    // DMA2
    *(int*)0xC61@io = 0;    // DMA3

    Peripheral_Reset();
}

/* ----- *
* C5535 REGISTERS *
* ----- */
#define ESCR      0x1c33

#define SDTIMR1   0x1020
#define SDTIMR2   0x1021
#define SDCR1     0x1008
#define SDCR2     0x1009
#define SDSRETR   0x103C
#define SDRCCR    0x100C

#define PRCR      0x1C05
#define PCGCR1    0x1c02
#define PCGCR2    0x1c03
#define PSRCR     0x1c04

#define CLKCFGL   0x1c1e
#define CCR2      0x1c1f
#define CGCR1     0x1c20
#define CGCR2     0x1c21
#define CGCR3     0x1c22
#define CGCR4     0x1c23
#define CCSSR     0x1c24
#define IVPD      0x0049

//
*****
*****

```

```

/* Memory map based on MP/MC value (assume MP/MC = 0).    */
c5535_MapInit() {
    GEL_MapOn();
    GEL_MapReset();

    /*Program Space*/

    /* DARAM */
    GEL_MapAdd(0x0000C0,0,0x001F40,1,1);    /* DARAM0 */
    GEL_MapAdd(0x002000,0,0x002000,1,1);    /* DARAM1 */
    GEL_MapAdd(0x004000,0,0x002000,1,1);    /* DARAM2 */
    GEL_MapAdd(0x006000,0,0x002000,1,1);    /* DARAM3 */
    GEL_MapAdd(0x008000,0,0x002000,1,1);    /* DARAM4 */
    GEL_MapAdd(0x00A000,0,0x002000,1,1);    /* DARAM5 */
    GEL_MapAdd(0x00C000,0,0x002000,1,1);    /* DARAM6 */
    GEL_MapAdd(0x00E000,0,0x002000,1,1);    /* DARAM7 */

    /* SARAM */
    GEL_MapAdd(0x010000,0,0x002000,1,1);    /* SARAM0 */
    GEL_MapAdd(0x012000,0,0x002000,1,1);    /* SARAM1 */
    GEL_MapAdd(0x014000,0,0x002000,1,1);    /* SARAM2 */
    GEL_MapAdd(0x016000,0,0x002000,1,1);    /* SARAM3 */
    GEL_MapAdd(0x018000,0,0x002000,1,1);    /* SARAM4 */
    GEL_MapAdd(0x01A000,0,0x002000,1,1);    /* SARAM5 */
    GEL_MapAdd(0x01C000,0,0x002000,1,1);    /* SARAM6 */
    GEL_MapAdd(0x01E000,0,0x002000,1,1);    /* SARAM7 */
    GEL_MapAdd(0x020000,0,0x002000,1,1);    /* SARAM8 */
    GEL_MapAdd(0x022000,0,0x002000,1,1);    /* SARAM9 */
    GEL_MapAdd(0x024000,0,0x002000,1,1);    /* SARAM10 */
    GEL_MapAdd(0x026000,0,0x002000,1,1);    /* SARAM11 */
    GEL_MapAdd(0x028000,0,0x002000,1,1);    /* SARAM12 */
    GEL_MapAdd(0x02A000,0,0x002000,1,1);    /* SARAM13 */
    GEL_MapAdd(0x02C000,0,0x002000,1,1);    /* SARAM14 */
    GEL_MapAdd(0x02E000,0,0x002000,1,1);    /* SARAM15 */
    GEL_MapAdd(0x030000,0,0x002000,1,1);    /* SARAM16 */
    GEL_MapAdd(0x032000,0,0x002000,1,1);    /* SARAM17 */
    GEL_MapAdd(0x034000,0,0x002000,1,1);    /* SARAM18 */
    GEL_MapAdd(0x036000,0,0x002000,1,1);    /* SARAM19 */
    GEL_MapAdd(0x038000,0,0x002000,1,1);    /* SARAM20 */
    GEL_MapAdd(0x03A000,0,0x002000,1,1);    /* SARAM21 */
    GEL_MapAdd(0x03C000,0,0x002000,1,1);    /* SARAM22 */
    GEL_MapAdd(0x03E000,0,0x002000,1,1);    /* SARAM23 */
    GEL_MapAdd(0x040000,0,0x002000,1,1);    /* SARAM24 */
    GEL_MapAdd(0x042000,0,0x002000,1,1);    /* SARAM25 */
    GEL_MapAdd(0x044000,0,0x002000,1,1);    /* SARAM26 */
    GEL_MapAdd(0x046000,0,0x002000,1,1);    /* SARAM27 */
    GEL_MapAdd(0x048000,0,0x002000,1,1);    /* SARAM28 */
    GEL_MapAdd(0x04A000,0,0x002000,1,1);    /* SARAM29 */
    GEL_MapAdd(0x04C000,0,0x002000,1,1);    /* SARAM30 */

```

```

GEL_MapAdd(0x04E000,0,0x002000,1,1);    /* SARAM31 */

/* External-Memory */
GEL_MapAdd(0x050000,0,0x7B0000,1,1);    /* External-SDRAM */
GEL_MapAdd(0x800000,0,0x400000,1,1);    /* External-Async */
GEL_MapAdd(0xC00000,0,0x200000,1,1);    /* External-Async */
GEL_MapAdd(0xE00000,0,0x100000,1,1);    /* External-Async */
GEL_MapAdd(0xF00000,0,0x0E0000,1,1);    /* External-Async */

/* ROM */
GEL_MapAdd(0xFE0000,0,0x008000,1,0);    /* SAROM0 */
GEL_MapAdd(0xFE8000,0,0x008000,1,0);    /* SAROM1 */
GEL_MapAdd(0xFF0000,0,0x008000,1,0);    /* SAROM2 */
GEL_MapAdd(0xFF8000,0,0x008000,1,0);    /* SAROM3 */

/* Data Space */

/* DARAM */
GEL_MapAdd(0x000000,1,0x000060,1,1);    /* MMRs */
GEL_MapAdd(0x000060,1,0x000FA0,1,1);    /* DARAM0 */
GEL_MapAdd(0x001000,1,0x001000,1,1);    /* DARAM1 */
GEL_MapAdd(0x002000,1,0x001000,1,1);    /* DARAM2 */
GEL_MapAdd(0x003000,1,0x001000,1,1);    /* DARAM3 */
GEL_MapAdd(0x004000,1,0x001000,1,1);    /* DARAM4 */
GEL_MapAdd(0x005000,1,0x001000,1,1);    /* DARAM5 */
GEL_MapAdd(0x006000,1,0x001000,1,1);    /* DARAM6 */
GEL_MapAdd(0x007000,1,0x001000,1,1);    /* DARAM7 */

/* SARAM */
GEL_MapAdd(0x008000,1,0x001000,1,1);    /* SARAM0 */
GEL_MapAdd(0x009000,1,0x001000,1,1);    /* SARAM1 */
GEL_MapAdd(0x00A000,1,0x001000,1,1);    /* SARAM2 */
GEL_MapAdd(0x00B000,1,0x001000,1,1);    /* SARAM3 */
GEL_MapAdd(0x00C000,1,0x001000,1,1);    /* SARAM4 */
GEL_MapAdd(0x00D000,1,0x001000,1,1);    /* SARAM5 */
GEL_MapAdd(0x00E000,1,0x001000,1,1);    /* SARAM6 */
GEL_MapAdd(0x00F000,1,0x001000,1,1);    /* SARAM7 */
GEL_MapAdd(0x010000,1,0x001000,1,1);    /* SARAM8 */
GEL_MapAdd(0x011000,1,0x001000,1,1);    /* SARAM9 */
GEL_MapAdd(0x012000,1,0x001000,1,1);    /* SARAM10 */
GEL_MapAdd(0x013000,1,0x001000,1,1);    /* SARAM11 */
GEL_MapAdd(0x014000,1,0x001000,1,1);    /* SARAM12 */
GEL_MapAdd(0x015000,1,0x001000,1,1);    /* SARAM13 */
GEL_MapAdd(0x016000,1,0x001000,1,1);    /* SARAM14 */
GEL_MapAdd(0x017000,1,0x001000,1,1);    /* SARAM15 */
GEL_MapAdd(0x018000,1,0x001000,1,1);    /* SARAM16 */
GEL_MapAdd(0x019000,1,0x001000,1,1);    /* SARAM17 */
GEL_MapAdd(0x01A000,1,0x001000,1,1);    /* SARAM18 */
GEL_MapAdd(0x01B000,1,0x001000,1,1);    /* SARAM19 */
GEL_MapAdd(0x01C000,1,0x001000,1,1);    /* SARAM20 */

```

```

GEL_MapAdd(0x01D000,1,0x001000,1,1);    /* SARAM21 */
GEL_MapAdd(0x01E000,1,0x001000,1,1);    /* SARAM22 */
GEL_MapAdd(0x01F000,1,0x001000,1,1);    /* SARAM23 */
GEL_MapAdd(0x020000,1,0x001000,1,1);    /* SARAM24 */
GEL_MapAdd(0x021000,1,0x001000,1,1);    /* SARAM25 */
GEL_MapAdd(0x022000,1,0x001000,1,1);    /* SARAM26 */
GEL_MapAdd(0x023000,1,0x001000,1,1);    /* SARAM27 */
GEL_MapAdd(0x024000,1,0x001000,1,1);    /* SARAM28 */
GEL_MapAdd(0x025000,1,0x001000,1,1);    /* SARAM29 */
GEL_MapAdd(0x026000,1,0x001000,1,1);    /* SARAM30 */
GEL_MapAdd(0x027000,1,0x001000,1,1);    /* SARAM31 */

/* External-Memory */
GEL_MapAdd(0x028000,1,0x3D8000,1,1);    /* External-SDRAM */
GEL_MapAdd(0x400000,1,0x200000,1,1);    /* External-Async */
GEL_MapAdd(0x600000,1,0x100000,1,1);    /* External-Async */
GEL_MapAdd(0x700000,1,0x080000,1,1);    /* External-Async */
GEL_MapAdd(0x780000,1,0x070000,1,1);    /* External-Async */

/* ROM */
GEL_MapAdd(0x7F0000,1,0x004000,1,0);    /* SAROM0 */
GEL_MapAdd(0x7F4000,1,0x004000,1,0);    /* SAROM1 */
GEL_MapAdd(0x7F8000,1,0x004000,1,0);    /* SAROM2 */
GEL_MapAdd(0x7FC000,1,0x004000,1,0);    /* SAROM3 */

/* IO Space */
GEL_MapAdd(0x0000,2,0xFFFF,1,1);        /* XPORT */
}

Peripheral_Reset()
{
    int i;

    *(short *)PSRCR@IO = 0x0020;
    *(short *)PRCR@IO  = 0x00BB;

    for(i=0;i<0xff;i++);
    *(short *)IVPD@data = 0x027F; // Load interrupt vector pointer
    GEL_TextOut("Reset Peripherals is complete.\n");
}

ProgramPLL_100MHz() {
    int i;

    GEL_TextOut("Configuring PLL (100 MHz).\n");
    /* Enable clocks to all peripherals */
    *(short *)PCGCR1@IO = 0x0;
    *(short *)PCGCR2@IO = 0x0;

    /* For 32KHz input clock */

```

```

/* Bypass PLL */
*(short *)CCR2@IO = 0x0;

/* Set CLR_CNTL = 0 */
*(short *)CGCR1@IO = *(short *)CGCR1@IO & 0x7FFF;

*(short *)CGCR1@IO = 0x8BE8;
*(short *)CGCR2@IO = 0x8000;
*(short *)CGCR3@IO = 0x0806;
*(short *)CGCR4@IO = 0x0000;

/* Wait for PLL lock */
for(i=0;i<0x7fff;i++);

/* Switch to PLL clk */
*(short *)CCR2@IO = 0x1;

GEL_TextOut("PLL Init Done.\n");
}

ProgramPLL_120MHz() {
    int i;

    GEL_TextOut("Configuring PLL (120 MHz).\n");
    /* Enable clocks to all peripherals */
    *(short *)PCGCR1@IO = 0x0;
    *(short *)PCGCR2@IO = 0x0;

    /* Bypass PLL */
    *(short *)CCR2@IO = 0x0;

    /* Set CLR_CNTL = 0 */
    *(short *)CGCR1@IO = *(short *)CGCR1@IO & 0x7FFF;

    *(short *)CGCR1@IO = 0x8E4B;
    *(short *)CGCR2@IO = 0x8000;
    *(short *)CGCR3@IO = 0x0806;
    *(short *)CGCR4@IO = 0x0000;

    /* Wait for PLL lock */
    for(i=0;i<0x7fff;i++);

    /* Switch to PLL clk */
    *(short *)CCR2@IO = 0x1;

    GEL_TextOut("PLL Init Done.");
}

```



```

ProgramPLL_60MHz() {
    int i;

    GEL_TextOut("Configuring PLL (60 MHz).\n");
    /* Enable clocks to all peripherals */
    *(short *)PCGCR1@IO = 0x0;
    *(short *)PCGCR2@IO = 0x0;

    /* Bypass PLL */
    *(short *)CCR2@IO = 0x0;

    /* Set CLR_CNTL = 0 */
    *(short *)CGCR1@IO = *(short *)CGCR1@IO & 0x7FFF;

    *(short *)CGCR1@IO = 0x8724;
    *(short *)CGCR2@IO = 0x8000;
    *(short *)CGCR3@IO = 0x0806;
    *(short *)CGCR4@IO = 0x0000;

    /* Wait for PLL lock */
    for(i=0;i<0x7fff;i++);

    /* Switch to PLL clk */
    *(short *)CCR2@IO = 0x1;

    GEL_TextOut("PLL Init Done.");
}

```

Memory Mapping

```

Linkx.cmd
/*****
*****/
/* LNKX.CMD - COMMAND FILE FOR LINKING C PROGRAMS IN LARGE/HUGE MEMORY
MODEL */
/* cl55 <src files> -z -o<out file> -m<map file> lnkx.cmd -l<RTS
library> */
/*****
*****/

/*-stack      0x2000      /* Primary stack size  */
/*-sysstack    0x1000      /* Secondary stack size */
/*-heap        0x2000      /* Heap area size      */

/* SPECIFY THE SYSTEM MEMORY MAP */

MEMORY
{

```

```

PAGE 0: /* ---- Unified Program/Data Address Space ---- */

    MMR      (RW) : origin = 0000000h length = 0000c0h /* MMRs */
    /*DARAM (RW)   : origin = 00000c0h length = 00ff40h*/ /* on-chip
DARAM */
    DARAM_0 (RW)  : origin = 00000c0h length = 001f40h
    DARAM_1 (RW)  : origin = 0002000h length = 004000h
    /*DARAM_2 (RW) : origin = 0004000h length = 002000h */
    DARAM_3 (RW)  : origin = 0006000h length = 002000h
    DARAM     (RW) : origin = 0008000h length = 008000h

    SARAM     (RW) : origin = 0010000h length = 040000h /* on-chip
SARAM */

    SAROM_0 (RX)  : origin = 0fe0000h length = 008000h      /* on-chip
ROM 0 */
    SAROM_1 (RX)  : origin = 0fe8000h length = 008000h      /* on-chip
ROM 1 */
    SAROM_2 (RX)  : origin = 0ff0000h length = 008000h      /* on-chip
ROM 2 */
    SAROM_3 (RX)  : origin = 0ff8000h length = 008000h      /* on-chip
ROM 3 */

    EMIF_CS0 (RW) : origin = 0050000h length = 07B0000h /* mSDR */
    EMIF_CS2 (RW) : origin = 0800000h length = 0400000h /* ASYNC1 :
NAND */
    EMIF_CS3 (RW) : origin = 0C00000h length = 0200000h /* ASYNC2 :
NAND */
    EMIF_CS4 (RW) : origin = 0E00000h length = 0100000h /* ASYNC3 :
NOR */
    EMIF_CS5 (RW) : origin = 0F00000h length = 00E0000h /* ASYNC4 :
SRAM */

PAGE 2: /* ----- 64K-word I/O Address Space ----- */

    IOPORT (RWI) : origin = 0x000000, length = 0x020000

}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */
SECTIONS
{
    vectors (NOLOAD)
    .bss      : > DARAM /*, fill = 0 */
    vector    : > DARAM      ALIGN = 256
    .stack    : > DARAM
    .sysstack : > DARAM
    .sysmem   : > DARAM
    .text     : > SARAM
    .data     : > DARAM

```

```

.cinit      : > DARAM
.const      : > DARAM
.cio        : > DARAM
.usect      : > DARAM
.switch     : > DARAM
.emif_cs0   : > EMIF_CS0
.emif_cs2   : > EMIF_CS2
.emif_cs3   : > EMIF_CS3
.emif_cs4   : > EMIF_CS4
.emif_cs5   : > EMIF_CS5

.ioport     > IOPORT PAGE 2          /* Global & static ioport vars
*/

/* --- MY DEFINITIONS --- */
/* For DSPLIB FFT */
/* .data:twiddle      : > DARAM_0 ALIGN = 2048 */
/* .fftwcode          : > SARAM */

cmplxBuf    : > DARAM_1
BufL        : > DARAM_1
BufR        : > DARAM_1
PSD         : > DARAM_1
PSD_sqrt    : > DARAM_1

tmpBuf      : > DARAM_1

brBuf       : > DARAM_1

wnd1        : > DARAM_3
wnd2        : > DARAM_3

rfftL       : > DARAM_1
ifftL       : > DARAM_1
rfftR       : > DARAM_1
ifftR       : > DARAM_1
}

/* C5535 HWAFFT ROM table addresses */
/* the HWAFFT does not work with a huge memory model, so this is not
being used */
_hwafft_br      = 0x00fefe9c;
_hwafft_8pts    = 0x00fefeb0;
_hwafft_16pts   = 0x00feff9f;
_hwafft_32pts   = 0x00ff00f5;
_hwafft_64pts   = 0x00ff03fe;
_hwafft_128pts  = 0x00ff0593;
_hwafft_256pts  = 0x00ff07a4;
_hwafft_512pts  = 0x00ff09a2;

```

```
_hwafft_1024pts = 0x00ff0c1c;
```