Dancing Fountain

Final Project Report
December 8, 2016
E155

Tess Despres and Hamza Khan

Abstract:

Visualizing music is interesting due to the complexity of different songs. This focus of this project was visualizing the beat of a song using a water fixture. This was done using a water fountain based oscillating metronome driven by a servo motor. The metronome fountain was designed to follow the beat of the music. The user first uses a python script to convert a .wav file to readable text. The system then takes in both the .wav file and .txt file and uses them to control the servo motor and speakers, respectively. In this way, the project is able to turn the fountain in beat with the music outputted from a speaker system.

Introduction:

This project used song data in order to create a water fountain metronome. In order to do this, the Raspberry Pi took in a .wav song file. A python script then converted the .wav file to .txt format. This information is then sent over SPI communication to an FPGA board. A 15 order low pass FIR filter, written on the FPGA, then filters the data. The information is then sent back over SPI to the Raspberry Pi. The Pi then uses a C algorithm to extract the beat from the filtered data set. The Pi then actuates a servo motor in time to the beat and plays the song from a second Raspberry Pi.

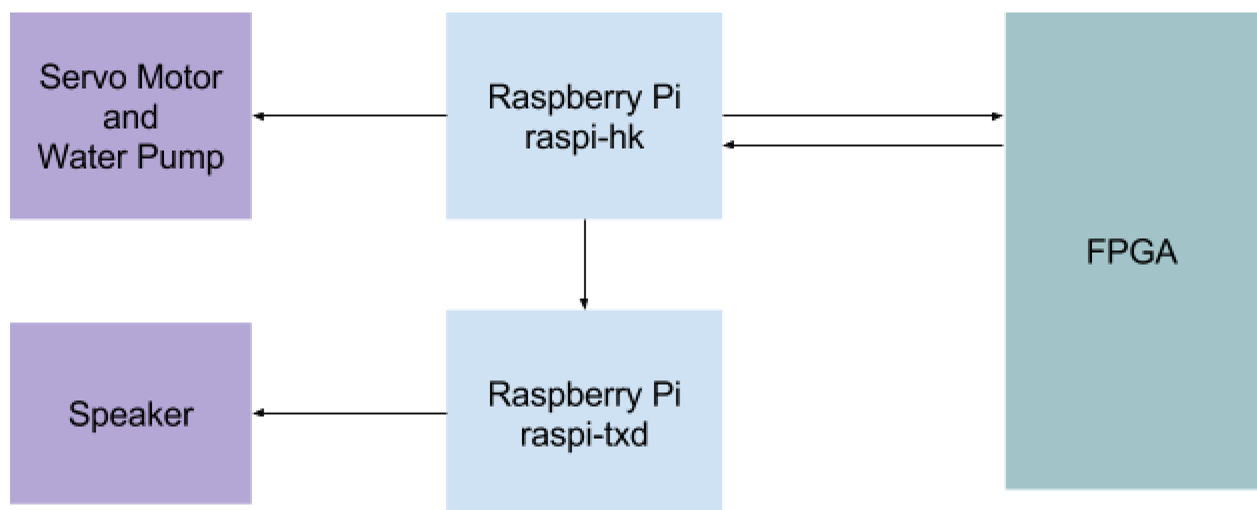

Figure 1: Block Diagram

This project was partitioned as shown in Figure 1. The FPGA handled SPI communication and FIR filtering. The Raspberry Pi did all other data processing and servo control. The Pi also handled communication using ssh to the second Pi which played the sound file.

New hardware:

This project incorporated three new hardware components. The first of these components is the servo motor. The motor used was a standard Hitec HS-311 motor with a 4.8V to 6.0V voltage range. This allowed the Pi to power and control the motor. This motor was controlled using the built in pulse width modulation peripheral. The peripheral used GPIO pin 18 to communicate with the motor.

The second hardware component incorporated was the water pump. The pump used was a SongLong SL-381 submersible pump with a flow rate of 80 GPH. This compact pump pushed water from a tupperware tank into the tubing connected to the servo motor. The water stream was oscillated back and forth by the servo motor.

The final extra hardware component that was included was external speakers. These speakers connected to both the aux plug on the second Raspberry Pi and a wall socket. This allowed the second Pi to play the song while the servo motor moved to the beat.

Schematics:

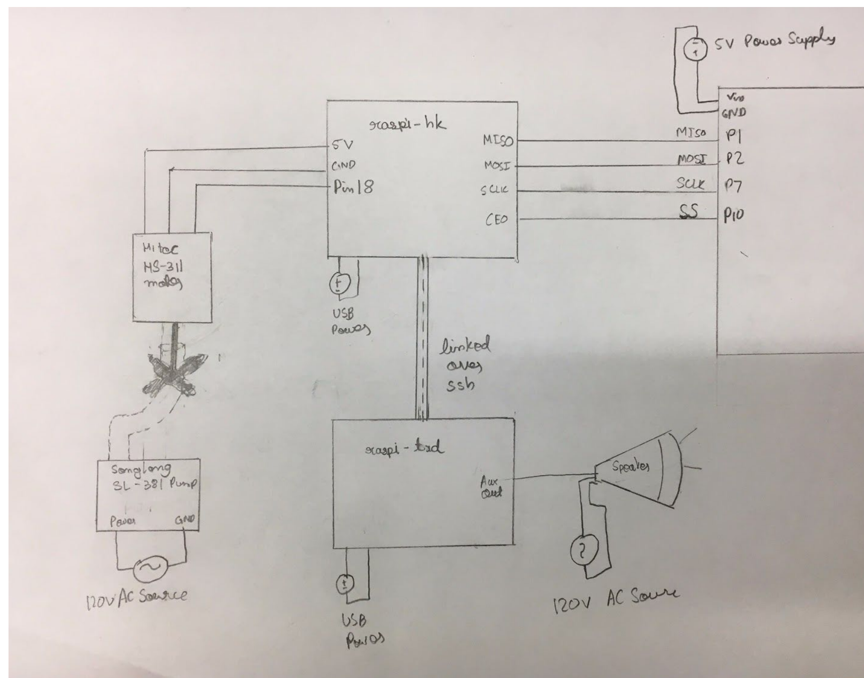

Figure 2: Overall System Schematic

The overall system schematic shows the electrical and physical connections in the system. Raspi-hk is the raspberry pi used to input the .wav file into the FPGA over SPI. This pi is also used to actuate the servo motor to oscillate 45 degrees off center to display the beat of the song. The servo motor is connected to a water tube which is shown on the schematic as a set of dashed lines. This raspberry pi also makes a secure shell connection with the other raspberry pi

(raspi-txd). This link is represented with a dashed line surrounded by 2 solid lines. The system uses the aux output of raspi-txd to play the song on a speaker.

Raspberry Pi Design:

The Raspberry Pi performed a number of tasks. These consisted of parsing the .wav files into .txt files, data processing. servo motor control, SPI communication protocol, ssh communication, and aux output. These tasks were spread out over two Raspberry Pi's. The first Raspberry Pi, raspi-hk, performed the conversion from .wav to .txt data on the song file, data processing, servo control, SPI communication, and sshed into the other Pi. The other Pi, raspi-txd, played the song out using the aux output.

In order to switch from .wav data to .txt data the Raspberry Pi ran a python script. This script parsed the data into array form using the wave python library. The data was converted by performing a 128 offset on the values greater than 64. The script then wrote this data into a .txt file.

The data processing performed on the Raspberry Pi occurred after the low pass processing on the FPGA. From the low passed data, it was evident, that beats occurred at large spikes with a magnitude of 65. Therefore, the algorithm took the numerical derivative using a current value pointer and previous value pointer. When the difference between these dereferenced values exceeded 60, the algorithm triggered a beat. In order to handle false positives in the data set, the algorithm also implemented a counter. It counts to 5000 which accounts for extra noise in the data set after the beat spike occurs.

The servo motor was controlled using pulse width modulation. The modulation used the built in pulse width modulation peripheral in the Raspberry Pi over GPIO 18. EasyPIO.h was used to initialize the peripherals. Additionally, a motor control and frequency function were written to control the pulses. The motor control function calls setPWM from EasyPIO.h in order to set the frequency to 50 Hz and duty cycle to 0.075. The frequency function controls the frequency of the servo motor oscillations by inputting a time delay of the period during each sweep of the motor. The time delay allows a maximum deflection of 45 degrees off center. If the period is too short to allow for a full sweep the servo motor will follow a shorter sweep due to switching direction more frequently.

SPI communication between the two Pi's was done using the sendrecieve function implemented in the library EasyPIO.h listed in *Appendix C* . The settings used to initialize the SPI transfer on the Pi for this system are such that the phase of the MOSI and MISO are aligned and data is transferred when SCLK is positive. Thus the settings input to initialize the SPI in the spiInit function is 0. The frequency used for the data transfer was 1kHz.

Ssh communication between the two Pi's were enabled using rsa encryption keys generated by raspi-hk. The main function in the C code, uses a system call to ssh into raspi-txd and plays the song. The built in function aplay was also called to play the song over the aux output. The aux output was connected to speakers which output the song. By using a fork call to start a new process the code was able to run both the servo control and song output code in parallel.

FPGA Design:

The FPGA was used to implement a low pass filter of cutoff frequency 20Hz. This was done to eliminate high frequency instruments and melodies in the song. Communication between the FPGA and the Raspberry Pi was done using SPI. The low pass filter implemented in the final project was a 15th order FIR filter. To calculate the coefficients for this filter, Matlab's FDATool was used to try different low pass filters. These coefficients were then scaled and rounded to fit the size of a signed byte because the input music signal also has a bit width of a byte.

The low pass filter was implemented as an FIR filter using D Flip-Flops to delay the signal by 15 time steps. These values were multiplied by their respective coefficients and added to get the filtered signal. The bit width of this output was truncated to a byte. The truncation was done by getting the 7 least significant bits and the most significant bit to keep the sign of the output. SPI communication was used to transfer the unfiltered and filtered song data back and forth. The SPI module started reading SPI after slave select was driven to low. The song data was input from the raspberry pi on the positive edge of the SPI clock and the filtered data was updated on the negative edge of the SPI clock. Therefore making the value correct when SPI reads it on the positive edge of the clock. The SPI module used was from Chapter 9 of the book Digital Design and Computer Architecture by Sarah L. Harris and David Money Harris. The low pass filter was running on the clock on the µMudd Mark IV board and the SPI transfer is running of the SCLK. To combine these 2 systems synchronizers are used and every byte of MISO and MOSI data is stored in a RAM. The resulting hardware of can be seen in *Appendix A*.

A filter recommended for future use is a 147 order low pass filter which gave better results in simulation. This low pass filter gives a steeper bode plot because of the higher order, making the frequency cutoff sharper. This filter was implemented in hardware using Altera's megafunction. Although this megafunction hardware worked in modelsim because the output file matched the results we expected from Matlab Simulations, it wasn't implemented in hardware with success and so a 15th order low pass filter was used instead.

Results:

The FIR low pass filter took in data over SPI to output a low pass filtered signal. The output from the 15th order FIR from the FPGA is not the same as the result from previously conducted Matlab simulations. This maybe because of the truncation to the low passed signal to a byte because currently the algorithm takes in the 7 least significant bits along with the most significant bit for the getting the correct sign. The output signal does however create pulses which correlate to the beat of the song when a derivative of the filtered signal is taken to extract the change in impulses.

The C algorithm was successful in finding the correct frequency most of the time. The algorithm was tested on five different songs Laudamus Te by Antonio Vivaldi, Africa by Toto, When I'm Sixty Four by The Beatles, Paradise City by Guns and Roses, and Born in the USA by Bruce Springsteen. The algorithm had the most success with Laudamus Te outputting an average frequency of 1.98 Hz while the actual beat is 1.93 Hz. The actual beat of these song are compared with the algorithms average output in Table 1. For most of the songs, the beat was followed even though shifts in frequency in When I'm Sixty Four and Paradise City. From the table it is clear that in, In Born in the USA the beat wasn't followed correctly. When listening to the song, however, it can be observed that the servo motor is keeping in time to the drum hits. The drums hits occur at approximately 2.02 Hz. This higher beat is likely the reason for the algorithm outputting a high frequency.

The servo motor outputted the correct frequency generated by the code. After a full song's length, there was some noticeable drifting in the starting position. This is due to our oscillation lengths being set by time and not position. Since this time varied with each updated frequency, this introduced some unknown error into the position. This was fixed by resetting the motor position on each run. In the future this would be something to take into consideration when implementing the oscillating servo motor code.

Song	**Actual Frequency (Hz)**	**Algorithm Average Frequency (Hz)**
Born in the USA	0.98	1.96
Africa	1.53	1.97
Paradise City	1.67 - 3.2	1.99
Vivaldi	1.93	1.98
When I’m Sixty Four	1.17 - 2.34	1.91

Table 1: Algorithm Output Frequencies

References:

EaspyPIO from Prof. Spencer’s page: http://pages.hmc.edu/mspencer/fa15/e155/files/EasyPIO.h
Servo Motor Data Sheet: http://users.ece.utexas.edu/~valvano/Datasheets/ServoHS311.pdf
Chapter 9 from Digital Design and Computer Architecture by Sarah L. Harris and David Money Harris: http://pages.hmc.edu/harris/class/e155/09_Ch%2009_online.pdf

Parts List:

Quantity	**Part**
2	Raspberry Pi 3
1	Hitec HS-311 motor
1	SongLong SL-381
1	μMudd Mark IV
1	Water Tubing
2	Plumbing L-Joints
1	Speaker

Appendix A: Hardware Design

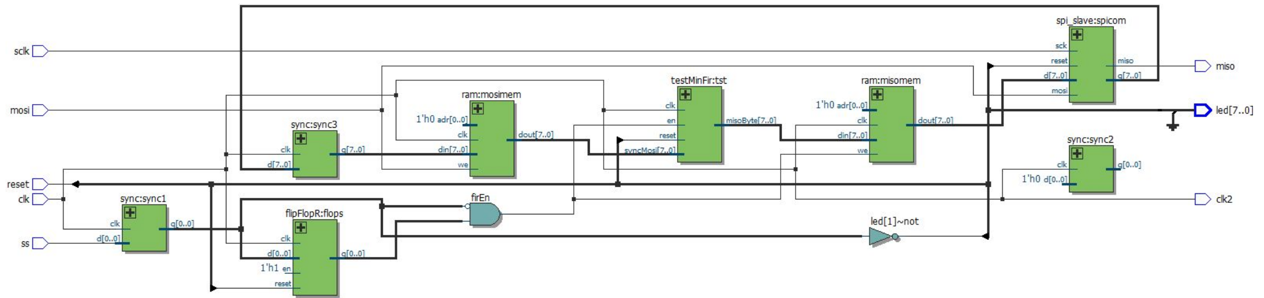

Figure 3: Top-level Hardware Schematic

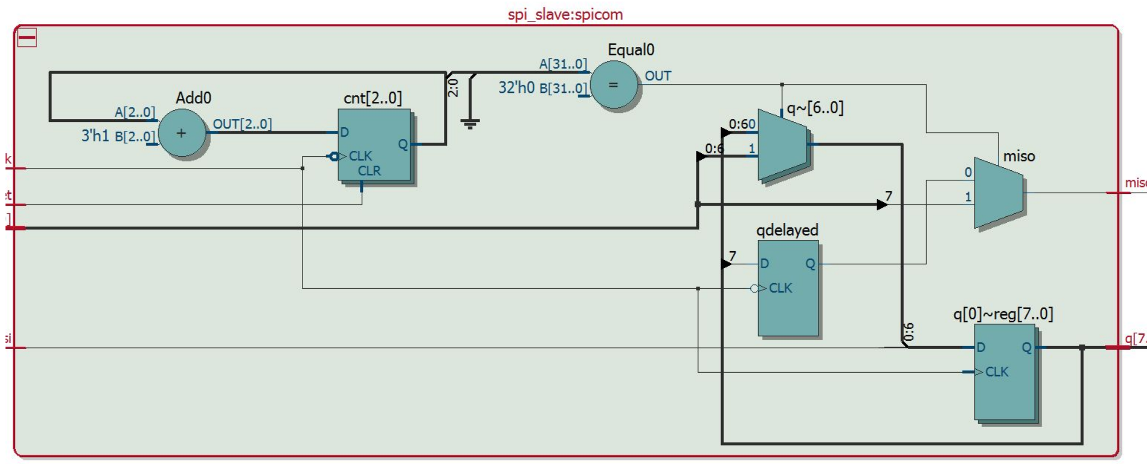

Figure 4: SPI Module

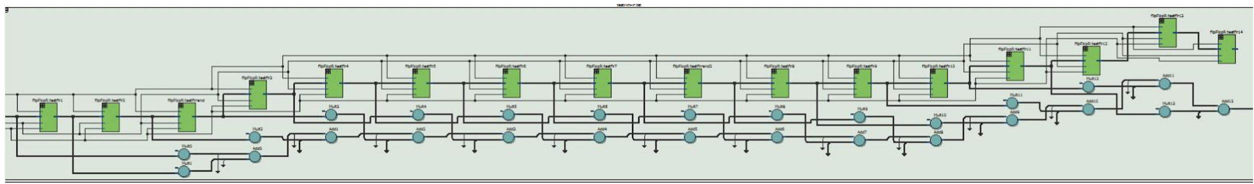

Figure 5: FIR Filter Module

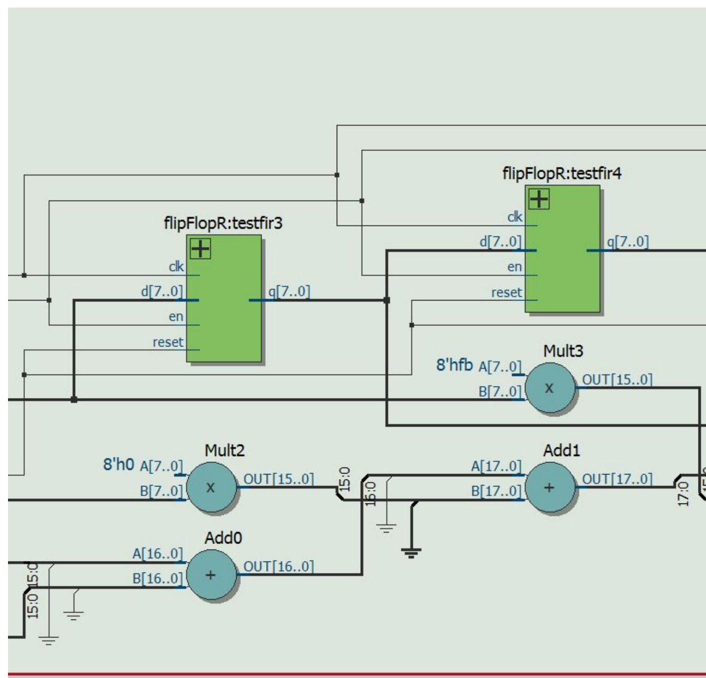

Figure 5: Zoomed-in view of the FIR filter

Appendix B: Verilog Code

```
//top level directory for SPI and FIR integration

module FinalTop( input logic clk,
                 input logic sclk,
                 input logic ss,
                 input logic reset,
                 input logic mosi,
                 output logic miso,
                 output logic clk2,
                 output logic[7:0] led);
        assign clk2 = clk;
        logic [7:0] mosiByte, syncMosi, misoByte, memMosi;
        logic byteUpdate, firEn, syncByteUpdate, syncSS, delayedSyncSS, mosiEn;
        logic tempReset;
        LEDs for testing
        assign led[0] = reset;
        assign led[1] = ~syncSS;
        //SPI data transfer module
        spi_slave spicom(sclk, mosi, miso, reset, misoByte, mosiByte);

        // Synchronizing the input signals
        sync #(1) sync1(clk, ss, syncSS);
        sync #(1) sync2(clk, byteUpdate, syncByteUpdate);
        sync #(8) sync3(clk, mosiByte, syncMosi);

        // Flip flop to time when the FIR module needs to be enabled
        flipFlopR #(1)flops(clk, reset, 1, syncSS, delayedSyncSS);
        assign firEn = delayedSyncSS&(~syncSS);
        assign mosiEn = (~delayedSyncSS)&syncSS;

        // Storing the MOSI value
        ram #(1,8) mosimem(clk, mosi, 0, syncMosi, memMosi);
```

```
		// 15th order low pass filter module
		testMinFir tst(clk, reset, firEn, memMosi, tempMiso);

		//Storing the MISO value
		ram #(1,8) misomem(clk, firEn, 0, tempMiso, misoByte);
		logic delaySS;
endmodule
```

```
// 15th order low pass filter module

module testMinFir(input logic clk, reset, en,
								input logic[7:0] syncMosi,
								output logic[7:0] misoByte);

		logic[7:0] q0, q1, q2, q3, q4 ,q5, q6, q7, q8, q9, q10, q11, q12, q13, q14;
		//Flip Flops to delay the input by a timestep so the signal can be multiplied to the
coefficients for low pass filtering
		flipFlopR #(8)testfir1(clk, reset, en, syncMosi, q0);
		flipFlopR #(8)testfir2(clk, reset, en, q0, q1);
		flipFlopR #(8)testfirrand(clk, reset, en, q1, q2);
		flipFlopR #(8)testfir3(clk, reset, en, q2, q3);
		flipFlopR #(8)testfir4(clk, reset, en, q3, q4);
		flipFlopR #(8)testfir5(clk, reset, en, q4 ,q5);
		flipFlopR #(8)testfir6(clk, reset, en, q5, q6);
		flipFlopR #(8)testfir7(clk, reset, en, q6, q7);
		flipFlopR #(8)testfirrand2(clk, reset, en, q7, q8);
		flipFlopR #(8)testfir8(clk, reset, en, q8, q9);
		flipFlopR #(8)testfir9(clk, reset, en, q9, q10);
		flipFlopR #(8)testfir10(clk, reset, en, q10, q11);
		flipFlopR #(8)testfir11(clk, reset, en, q11, q12);
		flipFlopR #(8)testfir12(clk, reset, en, q12, q13);
		flipFlopR #(8)testfir13(clk, reset, en, q13, q14);
		flipFlopR #(8)testfir14(clk, reset, en, q14, q15);
```

```
		logic[7:0] coef0,
coef1,coef2,coef3,coef4,coef5,coef6,coef7,coef8,coef9,coef10,coef11,coef12,coef13,coef14,coef
15;

		// Coeficients used for the low pass filtering
		assign coef0 = 8'shfe;
		assign coef1 = 8'sh00;
		assign coef2 = 8'sh03;
		assign coef3 = 8'sh00;
		assign coef4 = 8'shfb;
		assign coef5 = 8'sh00;
		assign coef6 = 8'sh10;
		assign coef7 = 8'sh1A;
		assign coef8 = 8'sh10;
		assign coef9 = 8'sh00;
		assign coef10 = 8'shfb;
		assign coef11 = 8'sh00;
		assign coef12 = 8'sh03;
		assign coef13 = 8'sh00;
		assign coef14 = 8'shfe;
		logic[27:0] tempSum;
		logic[17:0] result1, result2, result3, result4;
		assign tempSum = (coef1*syncMosi) + (coef2*q0) + (coef3*q1) + (coef4*q2) +
(coef5*q3) + (coef6*q4) + (coef7*q5) + (coef8*q6) + (coef9*q7) + (coef10*q8) + (coef11*q9) +
(coef12*q10) + (coef13*q11) + (coef14*q12);

		assign misoByte = {tempSum[27], tempSum[6:0]};

endmodule




// The following code was from Chapter 9 of Digital Design and Computer Architecture by Sarah
L. Harris and David Money Harris
module spi_slave(input logic sck, // From master
```

```
                        input logic mosi, // From master
                            output logic miso, // To master
                            input logic reset, // System reset
                            input logic [7:0] d, // Data to send
                            output logic [7:0] q); // Data received
    logic [2:0] cnt;
    logic qdelayed;
// 3-bit counter tracks when full byte is transmitted
always_ff @(negedge sck, posedge reset)
  if (reset) cnt = 0;
    else cnt = cnt + 3'b1;

// Loadable shift register
// Loads d at the start, shifts mosi into bottom on each step
always_ff @(posedge sck)
q <= (cnt == 0) ? {d[6:0], mosi} : {q[6:0], mosi};
// Align miso to falling edge of sck
// Load d at the start
always_ff @(negedge sck)
qdelayed = q[7];

assign miso = (cnt == 0) ? d[7] : qdelayed;

endmodule
```

Appendix C: Raspberry Pi code

Python converter from .wav to .txt

```
#converter.py
#python function for converting a .wav file to a text file which can be sent via spi
#Hamza Khan and Tess Despres -  Microps Final Project

import wave
import math
import struct

#file to import
fileName = 'paradiseCity.wav'

#open file and get number of frames
f = wave.open(fileName, 'r')
fileLength = f.getnframes()

#read frames and build array map
waveFile = f.readframes(fileLength)
waveFileArray = map(ord,list(waveFile))

#fix format
frame =[0]*fileLength
for i in range(fileLength) :
	frame[i] = waveFileArray[i]
	if (frame[i] > 64) :
		frame[i] = frame[i] - 128
	elif frame[i] == 128:
		frame[i] = 0
	else:
		frame[i] = frame[i]

#build string
waveString = ''
```

```
for i in range(fileLength):
        waveString += str(frame[i])
        waveString += '\n '

#output values to .txt file
textFileName = fileName[:-3] + 'txt'
textFile = open(textFileName, 'w')
textFile.write(waveString)
textFile.close()
f.close()
```

C code

```
// finalProjectSPI.c
// hkhan@hmc.edu, tdespres@hmc.edu 21 November 2016
//
// Send wav file to the FPGA over SPI
// recieve data from FPGA over SPI
// control servo and aux out


////////////////////////////////////////////////
// #includes
////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include "EasyPIO.h"
#include <math.h>

////////////////////////////////////////////////
// Function Prototypes
////////////////////////////////////////////////

void testLowPassSPI(char*, char*, char*, char*, char*, char*, char*);
void print16(char*);
```

```
void printall(char*, char*, char*, char*, char*, char*, char*);
void motorControl(float);
void initialize();
void update (int , int );
float thesholding ();

#define LOAD_PIN 23
#define DONE_PIN 24
#define songLength 2000000

//initialize global variables
signed char numberArray[songLength];
int buffer[windowLength];
float frequencyBuffer[songLength];
float lowpass[songLength];
int* currentValuePointer;
int* previousValuePointer;
int* indexPointer;
int* newSongInputPointer;
float average;
float stdDev;
float variance;
float deviationFactor;
int trigger;
int counter;
int fullCount;
int counterDelay;

/////////////////////////////////////////////
// Main Function
/////////////////////////////////////////////

void main() {

  //initialize frequency variable
  float frequency;
```

```
//load the song into an array
loadSong();

//initialize a buffer to load 0's
        initializeWindowZeroes();

        //initialize SPI
        pioInit();
        spiInit(244000, 0);

        // Load and done pins
        pinMode(LOAD_PIN, OUTPUT);
        pinMode(DONE_PIN, INPUT);

//initialize counter for while loop
        int i = 0;

        //give initial values to the counters that are used for thresholding
        deviationFactor = 3.3;
        trigger = 0;
        counter = 0;
        counterDelay = 5000;

        while (1) {
                //SPI control
                SPI0CSbits.TA = 1;
                signed char tempNew = spiSendReceive(numberArray[i]);
            SPI0CSbits.TA = 0;

    //update new and old values
                signed int new = (signed int) tempNew;
                new = new^4;
                int old = *indexPointer;
                update (old, new);
```

```
//load current frequency
		frequency = thesholding();

		//fill freq buffer and add freq bounds
		if ((i % 22050) == 0) {
			if (frequency < 0.6) {
				frequencyBuffer[i/22050] = 0.6;
			} else if (frequency > 10.0) {
				frequencyBuffer[i/22050] = 10.0;
			} else {
				frequencyBuffer[i/22050] = frequency;
	}

		}

		//break out of while at end of song
if (i > (songLength-1)) {
			break;
		}
		i++;
	}

	//calculate mean frequency
	int indexFreq;
	int freqCounter = 0;
	for ( indexFreq = 0; indexFreq < (songLength/22050); indexFreq++) {
		freqCounter += frequencyBuffer[indexFreq];
	}

	float mean = ((float)freqCounter)/((float)(songLength/22050));


	if (!fork()) {
		//enable song playing on raspi-txd
		system("sshpass -p tesspi ssh pi@172.28.70.172 aplay sixtyfour.wav");
	} else {
```

```
		//play frequency's and delay for speaker output to start
		delayMillis(2000)
		int j;
		for ( j =0; j < (songLength/22050); j++) {
			motorControl(frequencyBuffer[j]);
		}
	}

}

//////////////////////////////////////////////
// Windowing Functions
//////////////////////////////////////////////


void initializeWindow () {
	//point to initial values
	currentValuePointer = &buffer[windowLength/2];
	previousValuePointer = &buffer[(windowLength/2)-1];
	indexPointer = &buffer[0];
}


int initializeWindowZeroes () {
	//point to initial values
	currentValuePointer = &buffer[windowLength/2];
	previousValuePointer = &buffer[(windowLength/2)-1];
	indexPointer = &buffer[0];

	return 0;
}

void update (int old, int new) {
	int* endPointer = &buffer[windowLength-1];

	//move index pointer
```

```
	if (indexPointer == endPointer) {
		indexPointer = &buffer[0];
	} else {
		indexPointer++;
	}

	//move middle pointer
	if (currentValuePointer == endPointer) {
		currentValuePointer = &buffer[0];
		previousValuePointer = endPointer;
	} else {
		previousValuePointer = currentValuePointer;
		currentValuePointer++;
	}

	//update average
	float fnew = (float)new;
	float fold = (float)old;
	float deltaOld =  fold/((float)windowLength);
	float deltaNew =  fnew/((float)windowLength);
	float oldAverage = average;
	average += (fnew-fold)/(float)windowLength;

	*indexPointer = new;

}

float thesholding () {

	//if derivative is > 60 trigger signal
	float diff = *currentValuePointer - *previousValuePointer;
	if(diff > 60.0) {
		trigger = 1;
	}
	else{
		trigger = 0;
```

```
        }

    //perform counting delay
        if (counter == 0 && trigger == 1) {
                counter = counter +1;
                trigger = 0;
        }
        else if (counter > counterDelay && trigger == 1) {
                fullCount = counter;
                counter = 1;
                trigger = 0;
        }
        else if (counter > 0) {
                counter = counter + 1;
        }

    //reset if counter exceeds or output frequency
        if (fullCount == 0) {
                return 0;
        } else {
                return 11025.0/(float)fullCount;
        }
}

/////////////////////////////////////////////
// Function to load the song
/////////////////////////////////////////////
int loadSong()
{
    FILE *myFile;
    myFile = fopen("sixtyfour.txt", "r");

    //read file into array
    int i;

    if (myFile == NULL)
```

```
    {
      printf("Error Reading File\n");
      exit (0);
    }
    for (i = 0; i < songLength; i++)
    {
      fscanf(myFile, "%hi\n,", &numberArray[i] );

    }

    fclose(myFile);

    return 0;
}



///////////////////////////////////////////////////
// Servo Motor Control Functions
///////////////////////////////////////////////////

int freq(float frequency, int pin) {

    //initialize counter variable
        int i;

        //calculate period
    float period = 1.0/frequency;

        //delay oscillations by period length
        float cycleTime = period*1000.0; //convert from seconds to milliseconds

    //initialize number of cycles
        int numCycles;
    //always do at least one cycle
        if (cycleTime > 2000.0) {
```

```
                        numCycles = 1;
   //else do 2 seconds of cycles
          } else {
                        numCycles = 2000.0/cycleTime;
          }

          //oscillate for 2 seconds
   for(i = 0; i < numCycles; i++) {
                        setPWM(50, 0.05);
                        delayMillis(cycleTime);
                        setPWM(50, 0.1);
                        delayMillis(cycleTime);
   }

   return 0;
}


void motorControl(float frequency) {
          //set up pwm
   float pwmFreq = 50;
          //initialize io and pwm
   pioInit();
   pwmInit();

   //set pwm to center motor
   setPWM(50, 0.075);
   //call frequency function to oscillate motor
   freq(frequency, 18);
}




///////////////////////////////////////////////
// SPI read functions
```

```
/////////////////////////////////////////////

void spiRead() {
  printf("Reached main\n");
  char filteredMidAF[16];
  char filteredHighAF[16];
  char filteredLowAF[16];
  char filteredAF[16];

  pioInit();
  printf("Past pioInit\n");
  spiInit(244000, 0);
  printf("Past spiInit\n");

  // Load and done pins
  pinMode(LOAD_PIN, OUTPUT);
  pinMode(DONE_PIN, INPUT);
  printf("Past pinmodes\n");

  testLowPassSPI(midAF, highAF, lowAF, filteredMidAF, filteredHighAF, filteredLowAF,
filteredAF);
  printf("MidAF:          \n");
  printall(midAF, highAF, lowAF, filteredMidAF, filteredHighAF, filteredLowAF, filteredAF);


  return;
}


/////////////////////////////////////////////
// Functions
/////////////////////////////////////////////

void printall(char *midAF, char *highAF, char *lowAF, char *filteredMidAF, char
*filteredHighAF, char *filteredLowAF, char *filteredAF) {
  printf("MidAF:          ");  print16(midAF);        printf("\n");
```

```
  printf("FilteredMidAF:       ");  print16(filteredMidAF);   printf("\n");
  printf("HighAF:              ");  print16(highAF);          printf("\n");
  printf("FilteredHighAF:      ");  print16(filteredHighAF);  printf("\n");
  printf("LowAF:               ");  print16(lowAF);           printf("\n");
  printf("FilteredLowAF:       ");  print16(filteredLowAF);   printf("\n");
  printf("FilteredLast:       ");  print16(filteredAF);   printf("\n");

}

void testLowPassSPI(char *midAF, char *highAF, char *lowAF, char *filteredMidAF, char
*filteredHighAF, char *filteredLowAF, char *filteredAF) {

  printf("Reached main\n");

  int i;
  int ready;

  digitalWrite(LOAD_PIN, 1);
  printf("Reached main\n");

  for(i = 0; i < 16; i++) {
    filteredMidAF[i] = spiSendReceive(midAF[i]);
  }

  for(i = 0; i < 16; i++) {
    filteredHighAF[i] = spiSendReceive(highAF[i]);
  }

  for(i = 0; i < 16; i++) {
    filteredLowAF[i] = spiSendReceive(lowAF[i]);
  }

  printf("Reached main\n");

  digitalWrite(LOAD_PIN, 0);
```

```
}

void print16(char *text) {
  int i;

  for(i = 0; i < 16; i++) {
    printf("%02x ",text[i]);
  }
  printf("\n");
}
```

```
/*	EasyPIO.h
 *		Created:		8 October 2013
 *						Sarah_Lichtman@hmc.edu &
Joshua_Vasquez@hmc.edu
 *		Last Modified:	5 April 2014
 *						Sarah_Lichtman@hmc.edu &
Joshua_Vasquez@hmc.edu
 *		15 August 2014
 *		David_Harris@hmc.edu  (simplify pinMode)
 *
 *	Library to simplify memory access on Raspberry Pi (Broadcom BCM2835).
 *	Must be run with root permissions using sudo.
*/

#ifndef EASY_PIO_H
#define EASY_PIO_H

// Include statements
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
```

```
//////////////////////////////////////////////////////////////////////
// Constants
//////////////////////////////////////////////////////////////////////

// GPIO FSEL Types
#define INPUT  0
#define OUTPUT 1
#define ALT0   4
#define ALT1   5
#define ALT2   6
#define ALT3   7
#define ALT4   3
#define ALT5   2

// Clock Manager Bitfield offsets:
#define PWM_CLK_PASSWORD 0x5a000000
#define PWM_MASH 9
#define PWM_KILL 5
#define PWM_ENAB 4
#define PWM_SRC 0

// PWM Constants
#define PLL_FREQUENCY 500000000 // default PLLD value is 500 [MHz]
#define CM_FREQUENCY 25000000   // max pwm clk is 25 [MHz]
#define PLL_CLOCK_DIVISOR (PLL_FREQUENCY / CM_FREQUENCY)

//////////////////////////////////////////////////////////////////////
// Memory Map
//////////////////////////////////////////////////////////////////////

// These #define values are specific to the BCM2835, taken from "BCM2835 ARM Peripherals"
//#define BCM2835_PERI_BASE        0x20000000
// Updated to BCM2836 for Raspberry Pi 2.0 Fall 2015 dmh
#define BCM2835_PERI_BASE        0x3F000000
```

```
#define GPIO_BASE            (BCM2835_PERI_BASE + 0x200000)
#define UART_BASE                              (BCM2835_PERI_BASE + 0x201000)
#define SPI0_BASE                     (BCM2835_PERI_BASE + 0x204000)
#define PWM_BASE                      (BCM2835_PERI_BASE + 0x20c000)

#define SYS_TIMER_BASE                    (BCM2835_PERI_BASE + 0x3000)
#define ARM_TIMER_BASE                    (BCM2835_PERI_BASE + 0xB000)

#define CM_PWM_BASE           (BCM2835_PERI_BASE + 0x101000)

#define BLOCK_SIZE (4*1024)

// Pointers that will be memory mapped when pioInit() is called
volatile unsigned int *gpio; //pointer to base of gpio
volatile unsigned int *spi;  //pointer to base of spi registers
volatile unsigned int *pwm;

volatile unsigned int *sys_timer;
volatile unsigned int *arm_timer; // pointer to base of arm timer registers

volatile unsigned int *uart;
volatile unsigned int *cm_pwm;

///////////////////////////////////////////////////////////////////////////////
// GPIO Registers
///////////////////////////////////////////////////////////////////////////////

// Function Select
#define GPFSEL    ((volatile unsigned int *) (gpio + 0))
typedef struct
{
   unsigned FSEL0      : 3;
   unsigned FSEL1      : 3;
   unsigned FSEL2      : 3;
   unsigned FSEL3      : 3;
   unsigned FSEL4      : 3;
```

```
    unsigned FSEL5      : 3;
    unsigned FSEL6      : 3;
    unsigned FSEL7      : 3;
    unsigned FSEL8      : 3;
    unsigned FSEL9      : 3;
    unsigned            : 2;
}gpfsel0bits;
#define GPFSEL0bits (*(volatile gpfsel0bits*) (gpio + 0))
#define GPFSEL0 (*(volatile unsigned int*) (gpio + 0))

typedef struct
{
    unsigned FSEL10      : 3;
    unsigned FSEL11      : 3;
    unsigned FSEL12      : 3;
    unsigned FSEL13      : 3;
    unsigned FSEL14      : 3;
    unsigned FSEL15      : 3;
    unsigned FSEL16      : 3;
    unsigned FSEL17      : 3;
    unsigned FSEL18      : 3;
    unsigned FSEL19      : 3;
    unsigned             : 2;
}gpfsel1bits;
#define GPFSEL1bits (*(volatile gpfsel1bits*) (gpio + 1))
#define GPFSEL1 (*(volatile unsigned int*) (gpio + 1))

typedef struct
{
    unsigned FSEL20      : 3;
    unsigned FSEL21      : 3;
    unsigned FSEL22      : 3;
    unsigned FSEL23      : 3;
    unsigned FSEL24      : 3;
    unsigned FSEL25      : 3;
    unsigned FSEL26      : 3;
```

```
    unsigned FSEL27      : 3;
    unsigned FSEL28      : 3;
    unsigned FSEL29      : 3;
    unsigned             : 2;
}gpfsel2bits;
#define GPFSEL2bits (* (volatile gpfsel2bits*) (gpio + 2))
#define GPFSEL2 (* (volatile unsigned int *) (gpio + 2))

typedef struct
{
    unsigned FSEL30      : 3;
    unsigned FSEL31      : 3;
    unsigned FSEL32      : 3;
    unsigned FSEL33      : 3;
    unsigned FSEL34      : 3;
    unsigned FSEL35      : 3;
    unsigned FSEL36      : 3;
    unsigned FSEL37      : 3;
    unsigned FSEL38      : 3;
    unsigned FSEL39      : 3;
    unsigned             : 2;
}gpfsel3bits;
#define GPFSEL3bits (* (volatile gpfsel3bits*) (gpio + 3))
#define GPFSEL3 (* (volatile unsigned int *) (gpio + 3))


typedef struct
{
    unsigned FSEL40      : 3;
    unsigned FSEL41      : 3;
    unsigned FSEL42      : 3;
    unsigned FSEL43      : 3;
    unsigned FSEL44      : 3;
    unsigned FSEL45      : 3;
    unsigned FSEL46      : 3;
    unsigned FSEL47      : 3;
```

```
    unsigned FSEL48      : 3;
    unsigned FSEL49      : 3;
    unsigned             : 2;
}gpfsel4bits;
#define GPFSEL4bits (* (volatile gpfsel4bits*) (gpio + 4))
#define GPFSEL4 (* (volatile unsigned int *) (gpio + 4))

typedef struct
{
    unsigned FSEL50      : 3;
    unsigned FSEL51      : 3;
    unsigned FSEL52      : 3;
    unsigned FSEL53      : 3;
    unsigned             : 20;
}gpfsel5bits;
#define GPFSEL5bits (* (volatile gpfsel5bits*) (gpio + 5))
#define GPFSEL5 (* (volatile unsigned int *) (gpio + 5))

// Pin Output Select
#define GPSET    ((volatile unsigned int *) (gpio + 7))
typedef struct
{
    unsigned SET0       : 1;
    unsigned SET1       : 1;
    unsigned SET2       : 1;
    unsigned SET3       : 1;
    unsigned SET4       : 1;
    unsigned SET5       : 1;
    unsigned SET6       : 1;
    unsigned SET7       : 1;
    unsigned SET8       : 1;
    unsigned SET9       : 1;
    unsigned SET10      : 1;
    unsigned SET11      : 1;
    unsigned SET12      : 1;
    unsigned SET13      : 1;
```

```
    unsigned SET14      : 1;
    unsigned SET15      : 1;
    unsigned SET16      : 1;
    unsigned SET17      : 1;
    unsigned SET18      : 1;
    unsigned SET19      : 1;
    unsigned SET20      : 1;
    unsigned SET21      : 1;
    unsigned SET22      : 1;
    unsigned SET23      : 1;
    unsigned SET24      : 1;
    unsigned SET25      : 1;
    unsigned SET26      : 1;
    unsigned SET27      : 1;
    unsigned SET28      : 1;
    unsigned SET29      : 1;
    unsigned SET30      : 1;
    unsigned SET31      : 1;
}gpset0bits;
#define GPSET0bits (* (volatile gpset0bits*) (gpio + 7))
#define GPSET0 (* (volatile unsigned int *) (gpio + 7))

typedef struct
{
    unsigned SET32      : 1;
    unsigned SET33      : 1;
    unsigned SET34      : 1;
    unsigned SET35      : 1;
    unsigned SET36      : 1;
    unsigned SET37      : 1;
    unsigned SET38      : 1;
    unsigned SET39      : 1;
    unsigned SET40      : 1;
    unsigned SET41      : 1;
    unsigned SET42      : 1;
    unsigned SET43      : 1;
```

```
    unsigned SET44      : 1;
    unsigned SET45      : 1;
    unsigned SET46      : 1;
    unsigned SET47      : 1;
    unsigned SET48      : 1;
    unsigned SET49      : 1;
    unsigned SET50      : 1;
    unsigned SET51      : 1;
    unsigned SET52      : 1;
    unsigned SET53      : 1;
    unsigned            : 10;
}gpset1bits;
#define GPSET1bits (* (volatile gpset1bits*) (gpio + 8))
#define GPSET1 (* (volatile unsigned int *) (gpio + 8))

// Pin Output Clear
#define GPCLR    ((volatile unsigned int *) (gpio + 10))
typedef struct
{
    unsigned CLR0      : 1;
    unsigned CLR1      : 1;
    unsigned CLR2      : 1;
    unsigned CLR3      : 1;
    unsigned CLR4      : 1;
    unsigned CLR5      : 1;
    unsigned CLR6      : 1;
    unsigned CLR7      : 1;
    unsigned CLR8      : 1;
    unsigned CLR9      : 1;
    unsigned CLR10     : 1;
    unsigned CLR11     : 1;
    unsigned CLR12     : 1;
    unsigned CLR13     : 1;
    unsigned CLR14     : 1;
    unsigned CLR15     : 1;
    unsigned CLR16     : 1;
```

```
    unsigned CLR17      : 1;
    unsigned CLR18      : 1;
    unsigned CLR19      : 1;
    unsigned CLR20      : 1;
    unsigned CLR21      : 1;
    unsigned CLR22      : 1;
    unsigned CLR23      : 1;
    unsigned CLR24      : 1;
    unsigned CLR25      : 1;
    unsigned CLR26      : 1;
    unsigned CLR27      : 1;
    unsigned CLR28      : 1;
    unsigned CLR29      : 1;
    unsigned CLR30      : 1;
    unsigned CLR31      : 1;
}gpclr0bits;
#define GPCLR0bits (* (volatile gpclr0bits*) (gpio + 10))
#define GPCLR0 (* (volatile unsigned int *) (gpio + 10))

typedef struct
{
    unsigned CLR32       : 1;
    unsigned CLR33       : 1;
    unsigned CLR34       : 1;
    unsigned CLR35       : 1;
    unsigned CLR36       : 1;
    unsigned CLR37       : 1;
    unsigned CLR38       : 1;
    unsigned CLR39       : 1;
    unsigned CLR40       : 1;
    unsigned CLR41       : 1;
    unsigned CLR42       : 1;
    unsigned CLR43       : 1;
    unsigned CLR44       : 1;
    unsigned CLR45       : 1;
    unsigned CLR46       : 1;
```

```
    unsigned CLR47       : 1;
    unsigned CLR48       : 1;
    unsigned CLR49       : 1;
    unsigned CLR50       : 1;
    unsigned CLR51       : 1;
    unsigned CLR52       : 1;
    unsigned CLR53       : 1;
    unsigned             : 10;
}gpclr1bits;
#define GPCLR1bits (* (volatile gpclr1bits*) (gpio + 11))
#define GPCLR1 (* (volatile unsigned int *) (gpio + 11))

// Pin Level
#define GPLEV    ((volatile unsigned int *) (gpio + 13))
typedef struct
{
    unsigned LEV0       : 1;
    unsigned LEV1       : 1;
    unsigned LEV2       : 1;
    unsigned LEV3       : 1;
    unsigned LEV4       : 1;
    unsigned LEV5       : 1;
    unsigned LEV6       : 1;
    unsigned LEV7       : 1;
    unsigned LEV8       : 1;
    unsigned LEV9       : 1;
    unsigned LEV10      : 1;
    unsigned LEV11      : 1;
    unsigned LEV12      : 1;
    unsigned LEV13      : 1;
    unsigned LEV14      : 1;
    unsigned LEV15      : 1;
    unsigned LEV16      : 1;
    unsigned LEV17      : 1;
    unsigned LEV18      : 1;
    unsigned LEV19      : 1;
```

```
    unsigned LEV20      : 1;
    unsigned LEV21      : 1;
    unsigned LEV22      : 1;
    unsigned LEV23      : 1;
    unsigned LEV24      : 1;
    unsigned LEV25      : 1;
    unsigned LEV26      : 1;
    unsigned LEV27      : 1;
    unsigned LEV28      : 1;
    unsigned LEV29      : 1;
    unsigned LEV30      : 1;
    unsigned LEV31      : 1;
}gplev0bits;
#define GPLEV0bits (* (volatile gplev0bits*) (gpio + 13))
#define GPLEV0 (* (volatile unsigned int *) (gpio + 13))


typedef struct
{
    unsigned LEV32       : 1;
    unsigned LEV33       : 1;
    unsigned LEV34       : 1;
    unsigned LEV35       : 1;
    unsigned LEV36       : 1;
    unsigned LEV37       : 1;
    unsigned LEV38       : 1;
    unsigned LEV39       : 1;
    unsigned LEV40       : 1;
    unsigned LEV41       : 1;
    unsigned LEV42       : 1;
    unsigned LEV43       : 1;
    unsigned LEV44       : 1;
    unsigned LEV45       : 1;
    unsigned LEV46       : 1;
    unsigned LEV47       : 1;
    unsigned LEV48       : 1;
```

```
    unsigned LEV49      : 1;
    unsigned LEV50      : 1;
    unsigned LEV51      : 1;
    unsigned LEV52      : 1;
    unsigned LEV53      : 1;
    unsigned            : 10;
}gplev1bits;
#define GPLEV1bits (* (volatile gplev1bits*) (gpio + 14))
#define GPLEV1 (* (volatile unsigned int *) (gpio + 14))

//////////////////////////////////////////////////////////////////////
// SPI Registers
//////////////////////////////////////////////////////////////////////

typedef struct
{
        unsigned CS             :2;
        unsigned CPHA                   :1;
        unsigned CPOL                   :1;
        unsigned CLEAR                  :2;
        unsigned CSPOL                  :1;
        unsigned TA             :1;
        unsigned DMAEN                  :1;
        unsigned INTD                   :1;
        unsigned INTR                   :1;
        unsigned ADCS                   :1;
        unsigned REN                    :1;
        unsigned LEN                    :1;
        unsigned LMONO                  :1;
        unsigned TE_EN                  :1;
        unsigned DONE                   :1;
        unsigned RXD            :1;
        unsigned TXD            :1;
        unsigned RXR                    :1;
        unsigned RXF                    :1;
        unsigned CSPOL0         :1;
```

```
	unsigned CSPOL1	:1;
	unsigned CSPOL2	:1;
	unsigned DMA_LEN :1;
	unsigned LEN_LONG	:1;
	unsigned	:6;
}spi0csbits;
#define SPI0CSbits (* (volatile spi0csbits*) (spi + 0))
#define SPI0CS (* (volatile unsigned int *) (spi + 0))

#define SPI0FIFO (* (volatile unsigned int *) (spi + 1))
#define SPI0CLK (* (volatile unsigned int *) (spi + 2))
#define SPI0DLEN (* (volatile unsigned int *) (spi + 3))

/////////////////////////////////////////////////////////////////////
// System Timer Registers
/////////////////////////////////////////////////////////////////////

typedef struct
{
	unsigned M0	:1;
	unsigned M1	:1;
	unsigned M2	:1;
	unsigned M3	:1;
	unsigned	:28;
}sys_timer_csbits;
#define SYS_TIMER_CSbits (*(volatile sys_timer_csbits*) (sys_timer + 0))
#define SYS_TIMER_CS	(* (volatile unsigned int*)(sys_timer + 0))

#define SYS_TIMER_CLO	(* (volatile unsigned int*)(sys_timer + 1))
#define SYS_TIMER_CHI	(* (volatile unsigned int*)(sys_timer + 2))
#define SYS_TIMER_C0	(* (volatile unsigned int*)(sys_timer + 3))
#define SYS_TIMER_C1	(* (volatile unsigned int*)(sys_timer + 4))
#define SYS_TIMER_C2	(* (volatile unsigned int*)(sys_timer + 5))
#define SYS_TIMER_C3	(* (volatile unsigned int*)(sys_timer + 6))

/////////////////////////////////////////////////////////////////////
```

```
// ARM Interrupt Registers
////////////////////////////////////////////////////////////////////

#define IRQ_PENDING_BASIC (* (volatile unsigned int *) (arm_timer + 128))
#define IRQ_PENDING1 (* (volatile unsigned int *) (arm_timer + 129))
#define IRQ_PENDING2 (* (volatile unsigned int *) (arm_timer + 130))

#define IRQ_ENABLE1 (* (volatile unsigned int *) (arm_timer + 132))
#define IRQ_ENABLE2 (* (volatile unsigned int *) (arm_timer + 133))
#define IRQ_ENABLE_BASIC (* (volatile unsigned int *) (arm_timer + 134))
#define IRQ_DISABLE1 (* (volatile unsigned int *) (arm_timer + 135))
#define IRQ_DISABLE2 (* (volatile unsigned int *) (arm_timer + 136))
#define IRQ_DISABLE_BASIC (* (volatile unsigned int *) (arm_timer + 137))

////////////////////////////////////////////////////////////////////
// ARM Timer Registers
////////////////////////////////////////////////////////////////////

#define ARM_TIMER_LOAD (* (volatile unsigned int *) (arm_timer + 256))
//TODO: make timer control struct
#define ARM_TIMER_CONTROL  (* (volatile unsigned int *) (arm_timer + 258))
#define ARM_TIMER_IRQCLR (* (volatile unsigned int*) (arm_timer + 259))
#define ARM_TIMER_RAWIRQ (* (volatile unsigned int *) (arm_timer + 260))
#define ARM_TIMER_RELOAD (* (volatile unsigned int *) (arm_timer + 262))
#define ARM_TIMER_DIV (* (volatile unsigned int *) (arm_timer + 263))

////////////////////////////////////////////////////////////////////
// UART Registers
////////////////////////////////////////////////////////////////////

typedef struct
{
   unsigned DATA       : 8;
   unsigned FE         : 1;
   unsigned PE         : 1;
   unsigned BE         : 1;
```

```
   unsigned OE        : 1;
   unsigned           : 20;
} uart_drbits;
#define UART_DRbits (* (volatile uart_drbits*) (uart + 0))
#define UART_DR (*(volatile unsigned int *) (uart + 0))

typedef struct
{
   unsigned int CTS       : 1;
   unsigned int DSR       : 1;
   unsigned int DCD       : 1;
   unsigned int BUSY      : 1;
   unsigned int RXFE      : 1;
   unsigned int TXFF      : 1;
   unsigned int RXFF      : 1;
   unsigned int TXFE      : 1;
   unsigned int RI        : 1;
   unsigned int           : 24;
} uart_frbits;
#define UART_FRbits (*(volatile uart_frbits*) (uart + 6))
#define UART_FR (*(volatile unsigned int *) (uart + 6))

typedef struct
{
   unsigned int IBRD      : 16;
   unsigned int           : 16;
} uart_ibrdbits;
#define UART_IBRDbits   (*(volatile uart_ibrdbits*) (uart + 9))
#define UART_IBRD (*(volatile unsigned int *) (uart + 9))

typedef struct
{
   unsigned int FBRD      : 6;
   unsigned int           : 26;
} uart_fbrdbits;
#define UART_FBRDbits    (*(volatile uart_fbrdbits*) (uart + 10))
```

```
#define UART_FBRD (*(volatile unsigned int *) (uart + 10))

typedef struct
{
    unsigned int BRK        : 1;
    unsigned int PEN        : 1;
    unsigned int EPS        : 1;
    unsigned int STP2       : 1;
    unsigned int FEN        : 1;
    unsigned int WLEN       : 2;
    unsigned int SPS        : 1;
    unsigned int            : 24;
} uart_lcrhbits;
#define UART_LCRHbits (* (volatile uart_lcrhbits*) (uart + 11))
#define UART_LCRH (*(volatile unsigned int *) (uart + 11))

typedef struct
{
    unsigned int UARTEN     : 1;
    unsigned int SIREN      : 1;
    unsigned int SIRLP      : 1;
    unsigned int            : 4;
    unsigned int LBE        : 1;
    unsigned int TXE        : 1;
    unsigned int RXE        : 1;
    unsigned int DTR        : 1;
    unsigned int RTS        : 1;
    unsigned int OUT1       : 1;
    unsigned int OUT2       : 1;
    unsigned int RTSEN      : 1;
    unsigned int CTSEN      : 1;
    unsigned int            : 16;
} uart_crbits;
#define UART_CRbits (* (volatile uart_crbits*) (uart + 12))
#define UART_CR (*(volatile unsigned int *) (uart + 12))
```

```
typedef struct
{
    unsigned int RIRMIS      : 1;
    unsigned int CTSRMIS     : 1;
    unsigned int DCDRMIS     : 1;
    unsigned int DSRRMIS     : 1;
    unsigned int RXRIS       : 1;
    unsigned int TXRIS       : 1;
    unsigned int RTRIS       : 1;
    unsigned int FERIS       : 1;
    unsigned int PERIS       : 1;
    unsigned int BERIS       : 1;
    unsigned int OERIS       : 1;
    unsigned int             : 21;
} uart_risbits;
#define UART_RISbits (* (volatile uart_risbits*) (uart + 15))
#define UART_RIS (*(volatile unsigned int *) (uart + 15))

//////////////////////////////////////////////////////////////////////
// PWM Registers
//////////////////////////////////////////////////////////////////////

typedef struct
{
    unsigned PWEN1      :1;
    unsigned MODE1      :1;
    unsigned RPTL1      :1;
    unsigned SBIT1      :1;
    unsigned POLA1      :1;
    unsigned USEF1      :1;
    unsigned CLRF1      :1;
    unsigned MSEN1      :1;
    unsigned PWEN2      :1;
    unsigned MODE2      :1;
    unsigned RPTL2      :1;
```

```
    unsigned SBIT2      :1;
    unsigned POLA2      :1;
    unsigned USEF2      :1;
    unsigned            :1;
    unsigned MSEN2      :1;
    unsigned            :16;
} pwm_ctlbits;
#define PWM_CTLbits (* (volatile pwm_ctlbits *) (pwm + 0))
#define PWM_CTL (*(volatile unsigned int *) (pwm + 0))

#define PWM_RNG1 (*(volatile unsigned int *) (pwm + 4))
#define PWM_DAT1 (*(volatile unsigned int *)(pwm + 5))

///////////////////////////////////////////////////////////////////////
// Clock Manager Registers
///////////////////////////////////////////////////////////////////////

typedef struct
{
    unsigned SRC        :4;
    unsigned ENAB       :1;
    unsigned KILL       :1;
    unsigned            :1;
    unsigned BUSY       :1;
    unsigned FLIP       :1;
    unsigned MASH       :2;
    unsigned            :13;
    unsigned PASSWD     :8;
}cm_pwmctl_bits;
#define CM_PWMCTLbits (* (volatile cm_pwmctl_bits *) (cm_pwm + 40))
#define CM_PWMCTL (* (volatile unsigned int*) (cm_pwm + 40))

typedef struct
{
    unsigned DIVF       :12;
    unsigned DIVI       :12;
```

```
   unsigned PASSWD     :8;
} cm_pwmdivbits;
#define CM_PWMDIVbits (* (volatile cm_pwmdivbits *) (cm_pwm + 41))
#define CM_PWMDIV (*(volatile unsigned int *)(cm_pwm + 41))

////////////////////////////////////////////////////////////////////
// General Functions
////////////////////////////////////////////////////////////////////

// TODO: return error code instead of printing (mem_fd, reg_map)
void pioInit() {
        int  mem_fd;
        void *reg_map;

        // /dev/mem is a psuedo-driver for accessing memory in the Linux filesystem
        if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
              printf("can't open /dev/mem \n");
              exit(-1);
        }

        reg_map = mmap(
          NULL,             //Address at which to start local mapping (null means don't-care)
    BLOCK_SIZE,       //Size of mapped memory block
    PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
    MAP_SHARED,       // This program does not have exclusive access to this memory
    mem_fd,           // Map to /dev/mem
    GPIO_BASE);       // Offset to GPIO peripheral

        if (reg_map == MAP_FAILED) {
    printf("gpio mmap error %d\n", (int)reg_map);
    close(mem_fd);
    exit(-1);
  }

        gpio = (volatile unsigned *)reg_map;
```

```
reg_map = mmap(
        NULL,           //Address at which to start local mapping (null means don't-care)
  BLOCK_SIZE,       //Size of mapped memory block
  PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
  MAP_SHARED,       // This program does not have exclusive access to this memory
  mem_fd,          // Map to /dev/mem
  SPI0_BASE);       // Offset to SPI peripheral

if (reg_map == MAP_FAILED) {
  printf("spi mmap error %d\n", (int)reg_map);
  close(mem_fd);
  exit(-1);
}

spi = (volatile unsigned *)reg_map;

reg_map = mmap(
        NULL,           //Address at which to start local mapping (null means don't-care)
  BLOCK_SIZE,       //Size of mapped memory block
  PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
  MAP_SHARED,       // This program does not have exclusive access to this memory
  mem_fd,          // Map to /dev/mem
  PWM_BASE);       // Offset to PWM peripheral

if (reg_map == MAP_FAILED) {
  printf("pwm mmap error %d\n", (int)reg_map);
  close(mem_fd);
  exit(-1);
}

pwm = (volatile unsigned *)reg_map;

reg_map = mmap(
        NULL,           //Address at which to start local mapping (null means don't-care)
  BLOCK_SIZE,       //Size of mapped memory block
  PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
```

```
  MAP_SHARED,       // This program does not have exclusive access to this memory
  mem_fd,           // Map to /dev/mem
  SYS_TIMER_BASE);       // Offset to Timer peripheral

if (reg_map == MAP_FAILED) {
  printf("sys timer mmap error %d\n", (int)reg_map);
  close(mem_fd);
  exit(-1);
}

sys_timer = (volatile unsigned *)reg_map;

reg_map = mmap(
        NULL,             //Address at which to start local mapping (null means don't-care)
  BLOCK_SIZE,       //Size of mapped memory block
  PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
  MAP_SHARED,       // This program does not have exclusive access to this memory
  mem_fd,           // Map to /dev/mem
  ARM_TIMER_BASE);       // Offset to interrupts


if (reg_map == MAP_FAILED) {
  printf("arm timer mmap error %d\n", (int)reg_map);
  close(mem_fd);
  exit(-1);
}

arm_timer = (volatile unsigned *)reg_map;

reg_map = mmap(
        NULL,             //Address at which to start local mapping (null means don't-care)
  BLOCK_SIZE,       //Size of mapped memory block
  PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
  MAP_SHARED,       // This program does not have exclusive access to this memory
  mem_fd,           // Map to /dev/mem
  UART_BASE);       // Offset to UART peripheral
```

```
    if (reg_map == MAP_FAILED) {
      printf("uart mmap error %d\n", (int)reg_map);
      close(mem_fd);
      exit(-1);
    }

    uart = (volatile unsigned *)reg_map;

    reg_map = mmap(
            NULL,             //Address at which to start local mapping (null means don't-care)
      BLOCK_SIZE,       //Size of mapped memory block
      PROT_READ|PROT_WRITE,// Enable both reading and writing to the mapped memory
      MAP_SHARED,       // This program does not have exclusive access to this memory
      mem_fd,           // Map to /dev/mem
      CM_PWM_BASE);       // Offset to ARM timer peripheral

    if (reg_map == MAP_FAILED) {
      printf("cm_pwm mmap error %d\n", (int)reg_map);
      close(mem_fd);
      exit(-1);
    }

    cm_pwm = (volatile unsigned *)reg_map;
          close(mem_fd);
}

////////////////////////////////////////////////////////////////////////
// Interrupt Functions
////////////////////////////////////////////////////////////////////////

int irq1, irq2, irqbasic;

void noInterrupts(void) {
    //save current interrupts
    irq1 = IRQ_ENABLE1;
```

```
    irq2 = IRQ_ENABLE2;
    irqbasic = IRQ_ENABLE_BASIC;

    //disable interrupts
    IRQ_DISABLE1 = irq1;
    IRQ_DISABLE2 = irq2;
    IRQ_DISABLE_BASIC = irqbasic;
}

void interrupts(void) {
    if(IRQ_ENABLE1 == 0){ // if interrupts are disabled
    //restore interrupts
        IRQ_ENABLE1 = irq1;
        IRQ_ENABLE2 = irq2;
        IRQ_ENABLE_BASIC = irqbasic;
    }
}

////////////////////////////////////////////////////////////////////
// GPIO Functions
////////////////////////////////////////////////////////////////////

void pinMode(int pin, int function) {
    int reg      =  pin/10;
    int offset   = (pin%10)*3;
    GPFSEL[reg] &= ~((0b111 & ~function) << offset);
    GPFSEL[reg] |=  ((0b111 &  function) << offset);
}

void digitalWrite(int pin, int val) {
    int reg = pin / 32;
    int offset = pin % 32;

    if (val) GPSET[reg] = 1 << offset;
    else     GPCLR[reg] = 1 << offset;
}
```

```
int digitalRead(int pin) {
   int reg = pin / 32;
   int offset = pin % 32;

   return (GPLEV[reg] >> offset) & 0x00000001;
}

void pinsMode(int pins[], int numPins, int fxn) {
   int i;
   for(i=0; i<numPins; ++i) {
      pinMode(pins[i], fxn);
   }
}

void digitalWrites(int pins[], int numPins, int val) {
   int i;
   for(i=0; i<numPins; i++) {
      digitalWrite(pins[i], (val & 0x00000001));
      val = val >> 1;
   }
}

int digitalReads(int pins[], int numPins) {
   int i, val = digitalRead(pins[0]);

   for(i=1; i<numPins; i++) {
      val |= (digitalRead(pins[i]) << i);
   }
   return val;
}

////////////////////////////////////////////////////////////////////
// Timer Functions
////////////////////////////////////////////////////////////////////
```

```
}

// RPi timer peripheral clock is 1MHz.
// M0 and M3 are used by the GPU, so we must use M1 or M2

void delayMicros(int micros) {
   SYS_TIMER_C1 = SYS_TIMER_CLO + micros;   // set the compare register
   // 1000 clocks per millisecond
   SYS_TIMER_CSbits.M1 = 1;                 // reset match flag to 0
   while(SYS_TIMER_CSbits.M1 == 0);         // wait until the match flag is set
}

void delayMillis(int millis) {
   delayMicros(millis*1000);                // 1000 microseconds per millisecond
}

////////////////////////////////////////////////////////////////////
// SPI Functions
////////////////////////////////////////////////////////////////////

void spiInit(int freq, int settings) {
   //set GPIO 8 (CE), 9 (MISO), 10 (MOSI), 11 (SCLK) alt fxn 0 (SPI0)
   pinMode(8, ALT0);
   pinMode(9, ALT0);
   pinMode(10, ALT0);
   pinMode(11, ALT0);

   //Note: clock divisor will be rounded to the nearest power of 2
   SPI0CLK = 250000000/freq;   // set SPI clock to 250MHz / freq
   SPI0CS = settings;
   SPI0CSbits.TA = 1;          // turn SPI on with the "transfer active" bit
}

char spiSendReceive(char send){
   SPI0FIFO = send;            // send data to slave
   while(!SPI0CSbits.DONE);    // wait until SPI transmission complete
   return SPI0FIFO;            // return received data
}
```

```
short spiSendReceive16(short send) {
   short rec;
   SPI0CSbits.TA = 1;        // turn SPI on with the "transfer active" bit
   rec = spiSendReceive((send & 0xFF00) >> 8); // send data MSB first
   rec = (rec << 8) | spiSendReceive(send & 0xFF);
   SPI0CSbits.TA = 0;        // turn off SPI
   return rec;
}

////////////////////////////////////////////////////////////////////
// UART Functions
////////////////////////////////////////////////////////////////////

void uartInit(int baud) {
   uint fb = 12000000/baud; // 3 MHz UART clock

   pinMode(14, ALT0);
   pinMode(15, ALT0);
   UART_IBRD = fb >> 6;      // 6 Fract, 16 Int bits of BRD
   UART_FBRD = fb & 63;
   UART_LCRHbits.WLEN = 3;   // 8 Data, 1 Stop, 0 Parity, no FIFO, no Flow
   UART_CRbits.UARTEN = 1;   // Enable uart.
}

char getCharSerial(void) {
   while (UART_FRbits.RXFE);   // Wait until data is available.
   return UART_DRbits.DATA;        // Return char from serial port.
}


void putCharSerial(char c) {
   while (!UART_FRbits.TXFE);
   UART_DRbits.DATA = c;
}
```

```
////////////////////////////////////////////////////////////////////
// Pulse Width Modulation Functions
////////////////////////////////////////////////////////////////////

void pwmInit() {
   pinMode(18, ALT5);

   // Configure the clock manager to generate a 25 MHz PWM clock.
   // Documentation on the clock manager is missing in the datasheet
   // but found in "BCM2835 Audio and PWM Clocks" by G.J. van Loo 6 Feb 2013.
   // Maximum operating frequency of PWM clock is 25 MHz.
   // Writes to the clock manager registers require simultaneous writing
   // a "password" of 5A to the top bits to reduce the risk of accidental writes.

   CM_PWMCTL = 0; // Turn off PWM before changing
   CM_PWMCTL =  PWM_CLK_PASSWORD|0x20; // Turn off clock generator
   while(CM_PWMCTLbits.BUSY); // Wait for generator to stop
   CM_PWMCTL = PWM_CLK_PASSWORD|0x206; // Src = unfiltered 500 MHz CLKD
   CM_PWMDIV = PWM_CLK_PASSWORD|(PLL_CLOCK_DIVISOR << 12); // PWM Freq
= 25 MHz
   CM_PWMCTL = CM_PWMCTL|PWM_CLK_PASSWORD|0x10;    // Enable PWM clock
   while (!CM_PWMCTLbits.BUSY);    // Wait for generator to start
   PWM_CTLbits.MSEN1 = 1;  // Channel 1 in mark/space mode
   PWM_CTLbits.PWEN1 = 1;  // Enable pwm
}

/**
 * dut is a value between 0 and 1
 * freq is pwm frequency in Hz
 */
void setPWM(float freq, float dut) {
   PWM_RNG1 = (int)(CM_FREQUENCY / freq);
   PWM_DAT1 = (int)(dut * (CM_FREQUENCY / freq));
}

void analogWrite(int val) {
```

```
	setPWM(78125, val/255.0);
}

#endif
```