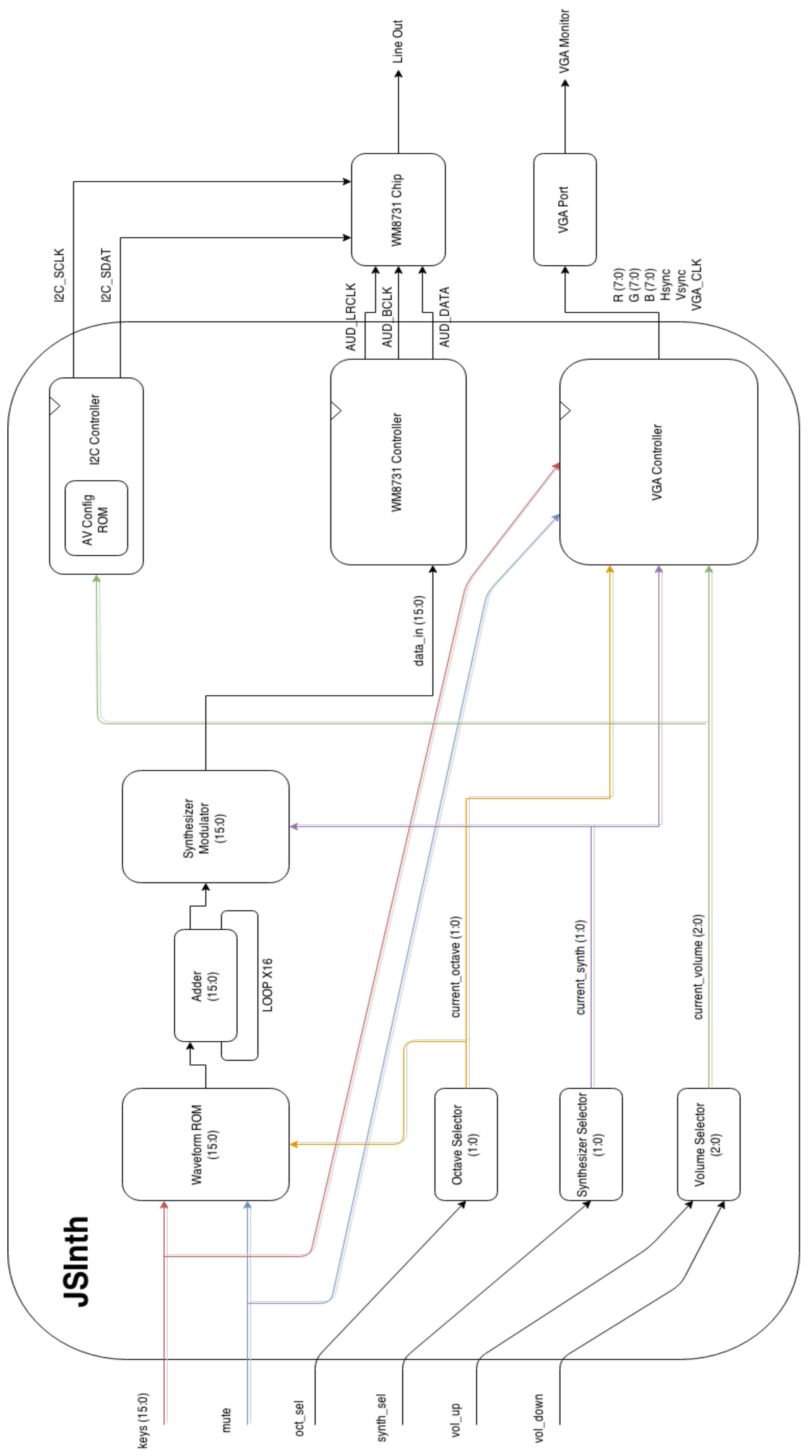
EECS 392: FPGA Systems Design Projects – Midterm Report

Introduction

Our goal for this design project is to build JS^{3}Inth, a 17-key synthesizer with multiple synth modulations and multi-state volume control implemented in real-time. It will be implemented on the Cyclone IV FPGA, and each switch, from left to right, will be mapped to a particular frequency corresponding to notes on a traditional keyboard, expanding through an octave and a half. We plan to implement JS^{3}Inth by creating VHDL entities that will take the inputs of which keys are turned on, get the corresponding notes from a ROM, add up the notes, amplify the result based on the current volume level, output this resultant sound to the speakers, and display information about the current state on the monitor.

Design Constraints and Requirements

The user should be able to cycle through several ranges of frequencies which represent octaves in a traditional keyboard. We will also allow the user to switch between multiple forms of sound modulation through a button input that will cycle through various programmed sound textures. We will also include a mute button that silences/un-silences the system. We will be also implementing a graphical user interface (GUI) which will be outputted through a video-graphics array (VGA), showing what keys are turned on, what volume level is being used, whether mute is on or off, and which synth modulation is being used. The only components/peripherals will be the Cyclone IV FPGA, the monitor, and the speakers.


JSInth
keys (15:0)
mute
oct_sel
synth_sel
vol_up
vol_down
Waveform ROM (15:0)
Adder (15:0)
LOOP X16
Synthesizer Modulator (15:0)
Octave Selector (1:0)
Synthesizer Selector (1:0)
Volume Selector (2:0)
current_octave (1:0)
current_synth (1:0)
current_volume (2:0)
data_in (15:0)
I2C Controller
AV Config ROM
WM8731 Controller
VGA Controller
I2C_SCLK
I2C_SDAT
AUD_LRCLK
AUD_BCLK
AUD_DATA
WM8731 Chip
Line Out
R (7:0)
G (7:0)
B (7:0)
Hsync
Vsync
VGA_CLK
VGA Port
VGA Monitor

Design Description

All inputs and outputs are sent through FIFO buffers to preserve the integrity of the signals. Initially, we have 16 switch inputs which are connected to a read-only memory (ROM). Each switch input will correspond to a point in memory which will then output a 16-bit sample to a frequency adder. Additionally, we will be multiplying the output of the frequency adder by a certain amount determined by the state of the octave state machine.

A finite state machine is implemented to determine the current synthesizer being used. Once a certain synth has been decided, the correct function inputs are pulled from the ROM and put into the synth modulation block, which combines with the frequency adder outputs to create a modified audio wave.

The volume is controlled through a finite state machine that cycles through 5 levels, these levels determine the maximum amplitude of the output signal. The output signal of the synth modulator is multiplied to increase the amplitude until it hits the correct volume. This signal is sent to the WM8731 controller to be serialized.

The I^{2}C controller controls the setup of the WM8731 chip using the standard I^{2}C protocol. The I^{2}C protocol works on a master-slave system using 2 data lines: Serial Clock and Serial Data. First, the master (In this case, the DE2-115 FPGA) pulls Serial Data low while Serial Clock remains high to initialize a transfer. The master then sends a signal of 8 bits on the data pin while cycling the clock. Then, the master clock cycles once and waits for acknowledgement of low from the slave, which will attempt to pull the Serial Data line low. After this acknowledgement bit, the master system will continue to shift in 8 bits on the Serial Data line and waiting for an acknowledgement bit from the slave system. Finally, the master will end a transmission by pulling Serial Data from low to high while the Serial Clock is high. In the case of the WM8731, we send a total of 3 bytes. The first is the address byte, which is standard across all I^{2}C systems. The second byte is the register address byte to tell the WM8731 which register to get ready to fill. The final byte tells the WM8731 what to fill the register with. In our case, the address byte for our chip is 0x34. Below is a table of the registers of the WM8731 and how we initialize them. The major things are that we set the sampling frequency to 48 kHz, initialize the DAC, and set the input sample length to 16 bits. For more information please refer to the WM8731 datasheet.

Register	Data
00	1A
02	1A
04	7B
06	7B
08	F8
0A	06
0C	00
0E	01
10	02
12	01

The WM8731 controller prepares the data to be sent to the WM8731 on-board chip. It takes the master clock and divides it into 3 separate clocks: the audio master clock (rated at 12.5 megahertz

(MHz)), the left-right clock, and the bit clock. The sampling frequency, set at 48 kilohertz (kHz), is determined by our left-right clock, or the amount of times a sample is processed by the audio chip per second. We constructed a parallel-to-serial converter to send the data one bit at a time to the audio chip. The audio chip takes the data, left-right clock and bit clock, and passes it through its Digital-to-Analog Converter (DAC), which connects to the Line Out component in the board, giving us sound.

All inputs are also passed through our VGA module. Our VGA controller contains rendering logic which draws colors and shapes on screen through a scanline, in which the synchronization module keeps track of the vertical and horizontal sync. The rendering logic checks every pixel the scanline is currently rendering, and draws a color correspondingly. Our controller reads the inputs and displays a simple user interface to help the user keep track of any of the keys, volumes, and octaves running in the system.

In order to produce a wave in ROM, we determine the wanted frequency and we extract the period in milliseconds. Due to the sampling rate, we sample 2 bytes of data every 30.82 microseconds. We create each angle in the wave by adding the cell's value by 360, dividing by the number of total samples. Then we calculate the sine of the angle and we multiply by 32767 (2^8). Converting these values to hexadecimal gives us the final value to be placed in our look-up table. Below is an example of values for the tone E5 with a frequency of 660 Hz.

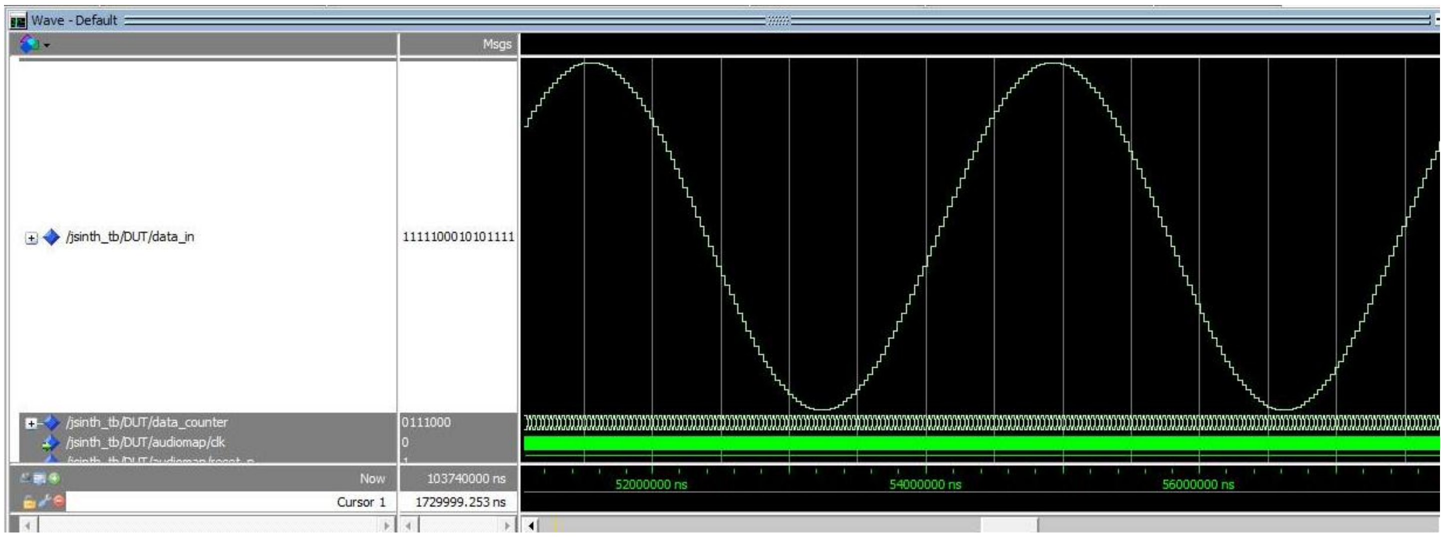
step # (decimal	Time (ms)	Angles	sin(Angle)	New Value (D	New Value (Hex)	Step # (binary)
0	0	0	0	0	0000	0000000
1	0.03082	7.346939	0.127877162	4190.150957	105E	0000001
2	0.06164	14.69388	0.253654584	8311.499751	2077	0000010
3	0.09246	22.04082	0.375267005	12296.37395	3008	0000011
4	0.12328	29.38776	0.490717552	16079.34203	3ECF	0000100
5	0.1541	36.73469	0.59811053	19598.28775	4C8E	0000101
6	0.18492	44.08163	0.695682551	22795.43014	590B	0000110
7	0.21574	51.42857	0.781831482	25618.27219	6412	0000111
8	0.24656	58.77551	0.855142763	28020.46292	6D74	0001000
9	0.27738	66.12245	0.914412623	29962.55842	750A	0001001
10	0.3082	73.46939	0.958667853	31412.66954	7AB4	0001010
11	0.33902	80.81633	0.987181783	32346.9855	7E5A	0001011
12	0.36984	88.16327	0.999486216	32750.16485	7FEE	0001100
13	0.40066	95.5102	0.995379113	32615.58739	7F67	0001101
14	0.43148	102.8571	0.974927912	31945.4629	7CC9	0001110
15	0.4623	110.2041	0.938468422	30750.79479	781E	0001111
16	0.49312	117.551	0.886599306	29051.19947	717B	0010000
17	0.52394	124.898	0.820172255	26874.58427	68FA	0010001
18	0.55476	132.2449	0.740277997	24256.68913	5EC0	0010010
19	0.58558	139.5918	0.648228395	21240.49983	52F8	0010011
20	0.6164	146.9388	0.545534901	17875.54211	45D3	0010100
21	0.64722	154.2857	0.433883739	14217.06848	3789	0010101
22	0.67804	161.6327	0.315108218	10325.15098	2855	0010110
23	0.70886	168.9796	0.191158629	6263.694787	1877	0010111
24	0.73968	176.3265	0.06407022	2099.388898	0833	0011000
25	0.7705	183.6735	-0.06407022	63434.6111	F7CA	0011001
26	0.80132	191.0204	-0.191158629	59270.30521	E786	0011010
27	0.83214	198.3673	-0.315108218	55208.84902	D7A8	0011011
28	0.86296	205.7143	-0.433883739	51316.93152	C874	0011100
29	0.89378	213.0612	-0.545534901	47658.45789	BA2A	0011101
30	0.9246	220.4082	-0.648228395	44293.50017	AD05	0011110
31	0.95542	227.7551	-0.740277997	41277.31087	A13D	0011111
32	0.98624	235.102	-0.820172255	38659.41573	9703	0100000
33	1.01706	242.449	-0.886599306	36482.80053	8E82	0100001
34	1.04788	249.7959	-0.938468422	34783.20521	87DF	0100010
35	1.0787	257.1429	-0.974927912	33588.5371	8334	0100011
36	1.10952	264.4898	-0.995379113	32918.41261	8096	0100100
37	1.14034	271.8367	-0.999486216	32783.83515	800F	0100101
38	1.17116	279.1837	-0.987181783	33187.0145	81A3	0100110
39	1.20198	286.5306	-0.958667853	34121.33046	8549	0100111
40	1.2328	293.8776	-0.914412623	35571.44158	8AF3	0101000
41	1.26362	301.2245	-0.855142763	37513.53708	9289	0101001
42	1.29444	308.5714	-0.781831482	39915.72781	9BEB	0101010
43	1.32526	315.9184	-0.695682551	42738.56986	A6F2	0101011
44	1.35608	323.2653	-0.59811053	45935.71225	B36F	0101100
45	1.3869	330.6122	-0.490717552	49454.65797	C12E	0101101
46	1.41772	337.9592	-0.375267005	53237.62605	CFF5	0101110
47	1.44854	345.3061	-0.253654584	57222.50025	DF86	0101111
48	1.47936	352.6531	-0.127877162	61343.84904	EF9F	0110000

Performance / Testing

Modules in our design were tested in ModelSim to ensure functionality. The I^2C module could not be simulated in ModelSim due to the need of the acknowledgement from the slave system (in this case the WM8731 chip). The finite state machines were tested by cycling through their states and ensuring they are encoded correctly and changing their outputs as appropriate. The VGA module was simulated to test the correct color output at a certain point in screen, but checking every single color is a tedious challenge, and many of the values remain the same because we are outputting blocks of color, so instead we synthesized the module into the board to ensure the rendering logic is working correctly.

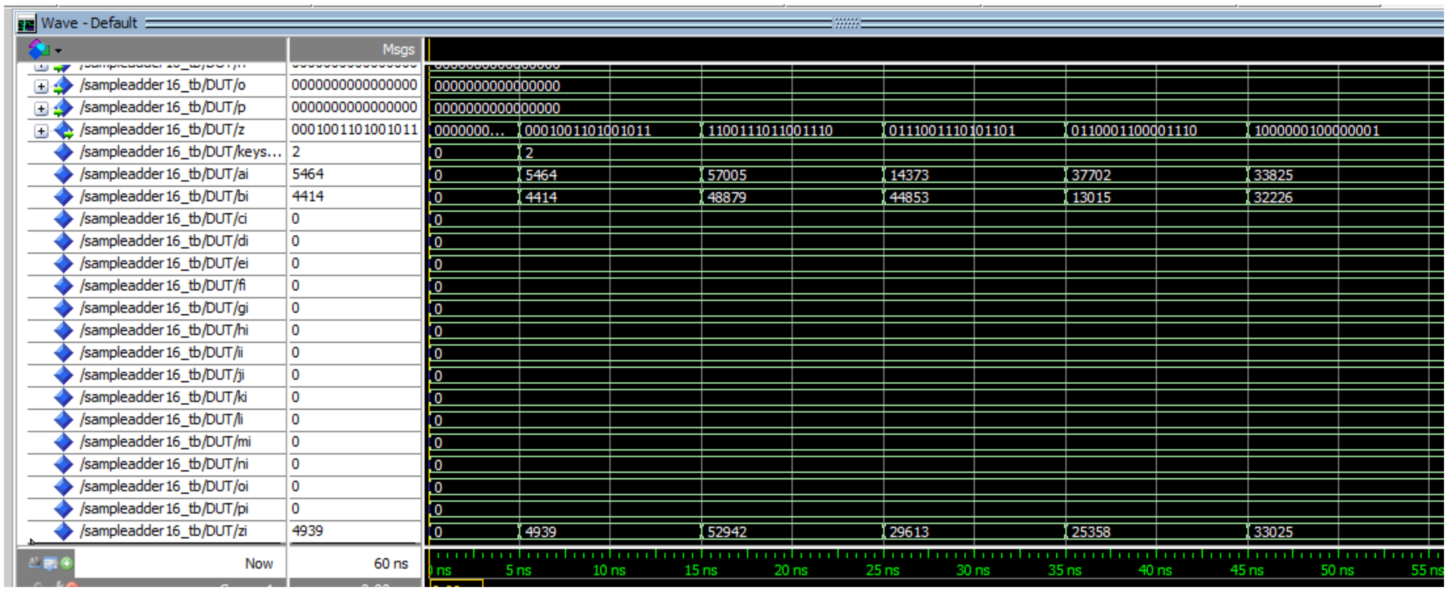
We simulated the WM8731 controller to ensure the module is outputting every signal wired to the WM8731 chip correctly. We simulated the bit clock to make sure the serializer worked correctly, the left-right clock to ensure the sampling rate met the correct constraint, and most importantly, we simulated our 16-bit data input to determine the correct wave is being sent correctly.

The biggest hurdle we encountered during development of the prototype of the JS^3Inth was figuring out how to get a digital audio signal out of the Altera DE2-115 successfully. This proved to be much more tedious than what we were originally anticipating since we were not sure on where to send our standard 16 bit audio signal in order to have it recognized and decoded by the on-board WM8731 audio codec properly. After scouring the DE2 user manual on how to set up the signal in the necessary serialized format, we eventually had to spend several days studying the datasheet for the Wolfson codec to learn how to communicate digitally with the chip as to set it up to receive and correctly understand the signal being sent to it through the I2C protocol.



```
Wave - Default
                                    Msgs
/jsinth_tb/DUT/data_in              1111100010101111
/jsinth_tb/DUT/data_counter         0111000
/jsinth_tb/DUT/audiomap/clk         0
                      Now           103740000 ns
                      Cursor 1      1729999.253 ns
```

16-bit Sample Wave Simulation



```
Wave - Default
                                    Msgs
/sampleadder16_tb/DUT/o             0000000000000000   0000000000000000
/sampleadder16_tb/DUT/p             0000000000000000   0000000000000000
/sampleadder16_tb/DUT/z             0001001101001011   0000000... 0001001101001011 1100111011001110 0111001110101101 0110001100001110 1000000100000001
/sampleadder16_tb/DUT/keys...       2                  0 2
/sampleadder16_tb/DUT/ai            5464               0 5464 57005 14373 37702 33825
/sampleadder16_tb/DUT/bi            4414               0 4414 48879 44853 13015 32226
/sampleadder16_tb/DUT/ci            0                  0
/sampleadder16_tb/DUT/di            0                  0
/sampleadder16_tb/DUT/ei            0                  0
/sampleadder16_tb/DUT/fi            0                  0
/sampleadder16_tb/DUT/gi            0                  0
/sampleadder16_tb/DUT/hi            0                  0
/sampleadder16_tb/DUT/ii            0                  0
/sampleadder16_tb/DUT/ji            0                  0
/sampleadder16_tb/DUT/ki            0                  0
/sampleadder16_tb/DUT/li            0                  0
/sampleadder16_tb/DUT/mi            0                  0
/sampleadder16_tb/DUT/ni            0                  0
/sampleadder16_tb/DUT/oi            0                  0
/sampleadder16_tb/DUT/pi            0                  0
/sampleadder16_tb/DUT/zi            4939               0 4939 52942 29613 25358 33025
                      Now           60 ns
```

16-bit Multi-Adder Simulation

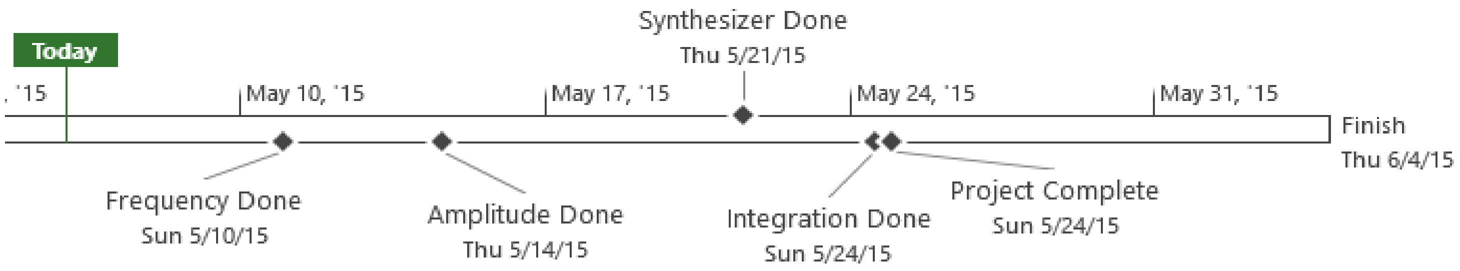
Development Plan

The next steps in our design is to implement, simulate and test the combination of different notes through chords. We will attempt to still remain with a 16-bit sample at combining chords, such that when we pass the output through our synth modulator, we will still be handling the same type of data.

We will attempt to re-write the I^{2}C controller to allow sending commands to the WM8731 to change the volume, instead of modifying the amplitude of the sample to control volume. This will optimize our design as we cut out a multiplied calculation for every bit.

We will also begin to write the master audio ROM to hold all our samples in the form of a lookup table; this will allow the keys to pull the value from the ROM parallelized, add and divide to form a signal, and pass this to our synth modulator.

The design so far is synthesized into the board and working correctly; connecting a VGA monitor and a pair of speakers allow us to use JSI^{3}nth at a very early build. The following is an updated Gantt Chart. We are at 58% completion of our design, and on schedule to finish by the end of Spring Quarter 2015.



Task Name	Duration	Start	Finish	Predecessors	Resource Names
Simulation + Small integration	**29.63 days**	**Fri 4/10/15**	**Thu 5/21/15**		
Theory	**10 days**	**Sun 4/12/15**	**Sun 4/26/15**		
Determine how the waveform should look	8 hrs	Sun 4/12/15	Thu 4/16/15		Jason, Spencer
Identify design bottlenecks	2 hrs	Sun 4/26/15	Sun 4/26/15		Jason, Spencer
VGA	**13.63 days**	**Sun 4/12/15**	**Thu 4/30/15**		
Determine how the screen output should look	5 hrs	Sun 4/12/15	Sun 4/12/15		Sebastian R.
VHDL	**10.13 days**	**Thu 4/16/15**	**Thu 4/30/15**	**6**	
Write Rendering Block	8 hrs	Thu 4/16/15	Sun 4/19/15		Sebastian R.
Write Sync Block	2 hrs	Thu 4/23/15	Thu 4/23/15		Sebastian R.
Write test bench	2 hrs	Sun 4/26/15	Sun 4/26/15	8,9	Sebastian R.
Testing + Debugging	6 hrs	Sun 4/26/15	Thu 4/30/15	10	Sebastian R.
VGA Done	0 days	Thu 4/30/15	Thu 4/30/15	11	Sebastian R.
Synthesizer	**28.63 days**	**Sun 4/12/15**	**Thu 5/21/15**		
Theory	**5 days**	**Sun 5/3/15**	**Sun 5/10/15**		
Determine Synth functions	8 hrs	Sun 5/3/15	Sun 5/10/15	3	Jason, Spencer
VHDL	**28.63 days**	**Sun 4/12/15**	**Thu 5/21/15**		
Write FIFO	4 hrs	Sun 4/12/15	Sun 4/12/15		Ian
Write Rom	2 hrs	Sun 5/3/15	Thu 5/7/15		Ian
Write Synthesizer	6 hrs	Sun 5/10/15	Sun 5/17/15	15	Ian
Write test bench	3 hrs	Sun 5/17/15	Sun 5/17/15	17,18,19	Ian
Testing + Debugging	4 hrs	Sun 5/17/15	Thu 5/21/15	20	Ian
Synthesizer Done	0 days	Thu 5/21/15	Thu 5/21/15	21	Ian
Amplitude + Frequency	**26 days**	**Fri 4/10/15**	**Sun 5/17/15**		
Amplitude Modulation	**24.88 days**	**Fri 4/10/15**	**Thu 5/14/15**		
Theory	**6 days**	**Fri 4/10/15**	**Sun 4/19/15**		

Determine how to amplify signal based on different volume	6 hrs	Fri 4/10/15	Sun 4/19/15		Jason, Spencer
VHDL	**23.88 days**	**Sun 4/12/15**	**Thu 5/14/15**		
Write Input	2 hrs	Sun 4/12/15	Sun 5/3/15		Sebastian R.
Write FIFO	3 hrs	Sun 4/19/15	Thu 4/23/15		Sebastian R.
Write Amplitude Modulation Block	4 hrs	Sun 5/3/15	Sun 5/10/15	26	Sebastian W.
Design testbench	4 hrs	Sun 5/10/15	Thu 5/14/15	28,29,30	Jason, Spencer
Testing + Debugging	3 hrs	Thu 5/14/15	Thu 5/14/15	31	Sebastian W.
Amplitude Done	0 days	Thu 5/14/15	Thu 5/14/15	32	Sebastian W.
Frequency Adder	**20 days**	**Thu 4/16/15**	**Thu 5/14/15**		
Theory	**0.38 days**	**Thu 4/23/15**	**Thu 4/23/15**		
Determine how to add signals	4 hrs	Thu 4/23/15	Thu 4/23/15		Jason, Spencer
VHDL	**20 days**	**Thu 4/16/15**	**Thu 5/14/15**		
Write Inputs	2 hrs	Thu 4/16/15	Thu 4/16/15		Ian
Write FIFO	3 hrs	Sun 4/19/15	Sun 4/19/15		Ian
Write ROM	3 hrs	Sun 4/19/15	Sun 4/19/15		Ian
Write Frequency Adder block	3 hrs	Sun 5/3/15	Sun 5/10/15	36	Sebastian W.
Design Test bench	2 hrs	Thu 5/14/15	Thu 5/14/15	38,39,40,41	Jason, Spencer
Testing + Debugging	3 hrs	Sun 5/3/15	Sun 5/10/15		Sebastian W.
Frequency Done	0 days	Sun 5/10/15	Sun 5/10/15	43	Sebastian W.
Output	**25 days**	**Sun 4/12/15**	**Sun 5/17/15**		
Research how to output audio	6 hrs	Sun 4/12/15	Sun 4/12/15		Sebastian W.
Write Audio out block	2 hrs	Thu 4/16/15	Thu 4/16/15	46	Sebastian W.
Write Test bench	2 hrs	Sun 5/3/15	Sun 5/10/15	47	Sebastian W.
Testing + Debugging	4 hrs	Sun 4/19/15	Sun 5/17/15	48	Sebastian W.
Audio Output Done	0 days	Sun 5/17/15	Sun 5/17/15	49	Sebastian W.

Final Integration	**1.38 days**	**Thu 5/21/15**	**Sun 5/24/15**	**1**	
Write Test bench	2 hrs	Thu 5/21/15	Thu 5/21/15		Jason, Spencer
Testing + Debugging	2 hrs	Sun 5/24/15	Sun 5/24/15	52	Ian, Jason, Sebastian W., Sebastian R., Spencer
Integration Done	0 days	Sun 5/24/15	Sun 5/24/15	53	
Implementation	**0 days**	**Sun 5/24/15**	**Sun 5/24/15**	**51**	
Load code onto FPGA	1 hr	Sun 5/24/15	Sun 5/24/15		Ian, Jason, Sebastian W., Sebastian R., Spencer
Debug	2 hrs	Sun 5/24/15	Sun 5/24/15	56	Ian, Jason, Sebastian W., Sebastian R., Spencer
Project Complete	0 days	Sun 5/24/15	Sun 5/24/15	57	
Other	**20 days**	**Thu 5/7/15**	**Thu 6/4/15**		
Documentation	**14.88 days**	**Thu 5/7/15**	**Thu 5/28/15**		
Write Project Report	2 hrs	Thu 5/7/15	Thu 5/7/15	51	Ian, Jason, Sebastian W., Sebastian R., Spencer
Write Final Report	2 hrs	Sun 5/24/15	Thu 5/28/15	55	Ian, Jason, Sebastian W., Sebastian R., Spencer
Major Milestones	**20 days**	**Thu 5/7/15**	**Thu 6/4/15**		
Project Report	0 days	Thu 5/7/15	Thu 5/7/15		
Final Report + Demo	0 days	Thu 6/4/15	Thu 6/4/15		