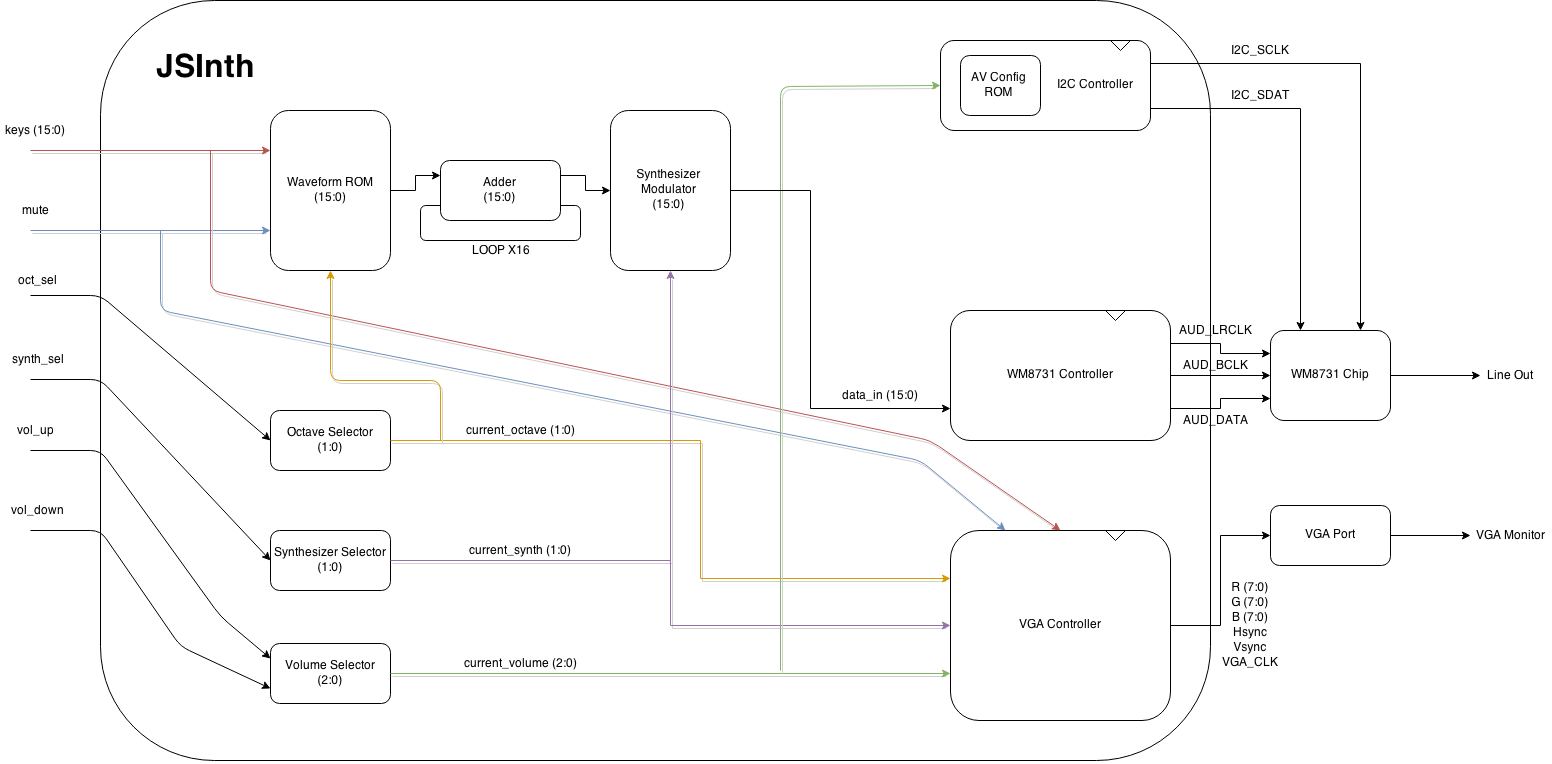
**EECS 392: FPGA Systems Design Projects – Midterm Report**

**Introduction**

Our goal for this design project is to build JS3Inth, 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 JS3Inth 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.



**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 I2C controller controls the setup of the WM8731 chip using the standard I2C protocol. The I2C 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 I2C 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 (28). 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.

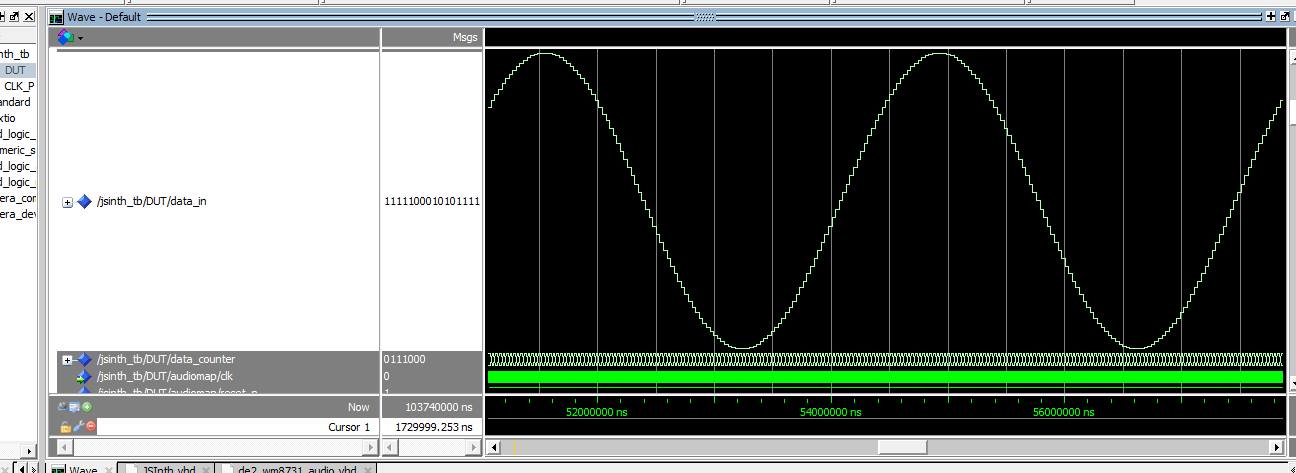


**Performance / Testing**

Modules in our design were tested in ModelSim to ensure functionality. The I2C 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 JS3Inth 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.



16-bit Sample Wave Simulation



16-bit Multi-Adder Simulation

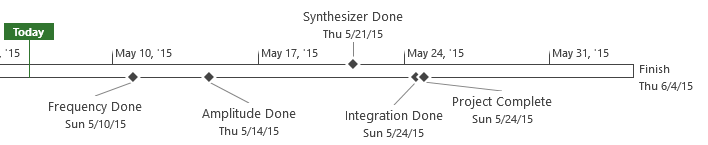
**Development Plan**

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 I2C 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 JSI3nth 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 |  |  |