

MUSIC COMPOSITION FOR DUMMIES

6.111 FINAL PROJECT REPORT

*By Wu, Yun
and
Seow, Shi Ling*

*6.111 (Spring 2005) – Introductory Digital Systems Laboratory
TA: Kehoe, Charlie
Date: May 12, 2005*

Abstract

For those who enjoy singing, but cannot read or write music, or for those who wish to compose more efficiently, this system allows a person to play or sing a simple tune and obtain a corresponding piece of sheet music on a video output. The system is made of two main parts: a pitch-detecting module and a video output module. The pitch detector determines the note being played based on the frequency of the input, which allows the video component to correctly draw the note being played onto a staff display. This proof-of-concept apparatus allows frequency detection of input sound waves, musical note determination, and display of notes onto a staff.

1. Design Overview

Our project “Music Composition for Dummies” allows amateurs to create a “masterpiece” by simply singing into a microphone. The microphone is inputted into a system that has two main parts: a pitch detector and a video display. The pitch detector retrieves audio data bits from an audio codec, performs a 1024-point Fast Fourier Transform, and determines the frequency and the note value of the input sound. This note value is fed into the video display module, which displays the note onto a staff on the video screen. Figure 1 shows the block diagram of the overall system.

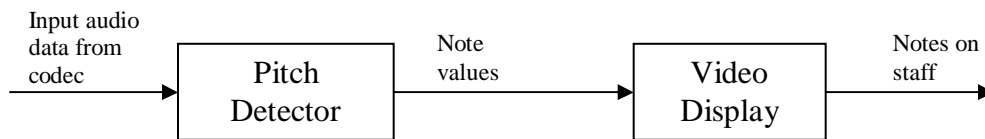


Figure 1: Block diagram of overall system

The project not only demonstrates how to extract audio data, how to use FFT for frequency detection, and how to implement a video display, but also incorporates many important aspects of large-scale digital systems design and integration. The entire system requires utilizing existing Xilinx audio, video, and labkit modules as well as developing finite state machines to correctly time and send control signals to the existing modules. The project is interesting because besides coding, it also allows us to build analog circuitry, use external peripherals, and employ our creativity while developing a product that is very practical for musicians and dummies alike. Our project demonstrates a proof-of-concept that the frequency of sound waves can be detected and displayed as musical notes. On a more robust level, our device could potentially allow anyone to easily compose music by singing, as well as play an existing song on a CD and obtain the corresponding score.

2. Design Description

2.1 Pitch Detector (Yun Wu)

The main purpose of the pitch detector is to transform audio data bits from the codec into a corresponding note value by performing a Fast Fourier Transform of the input data and determining the peak of the frequency domain audio spectrum. Figure 2 shows a block diagram of the pitch detector.

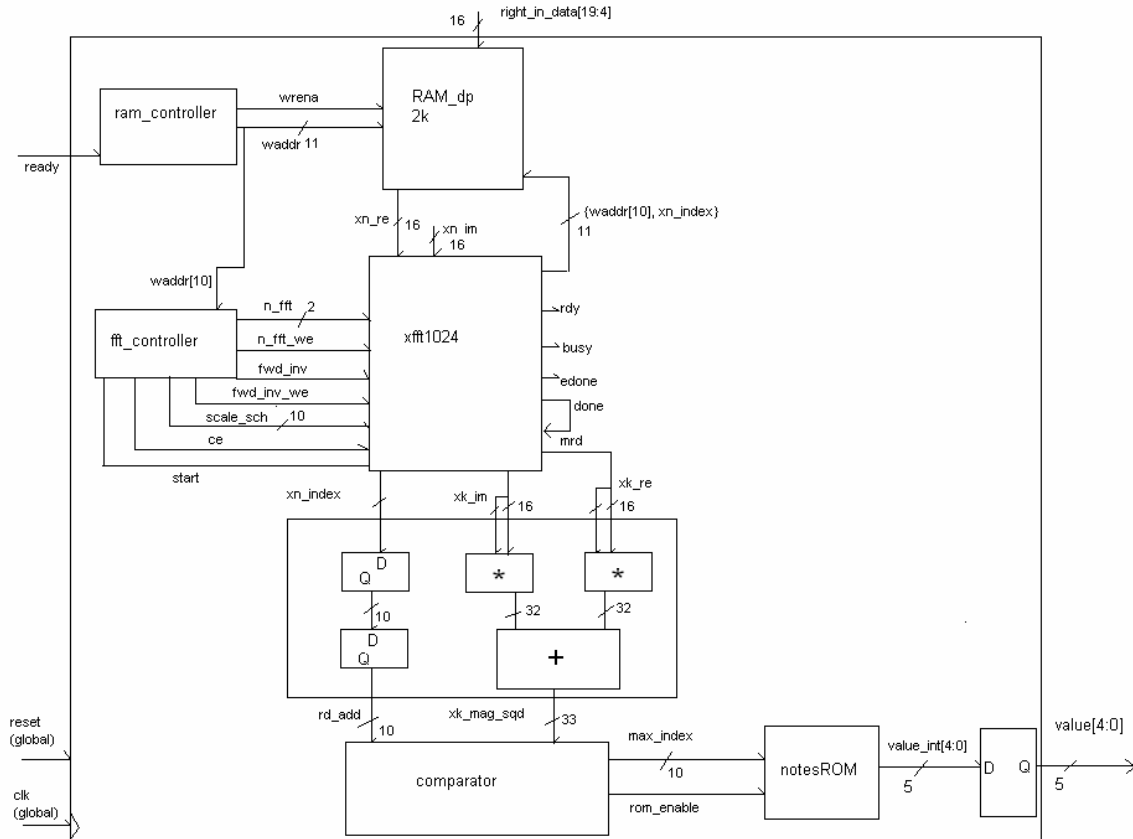


Figure 2: Block diagram of pitch detector

The external input signals of the pitch detector include a reset from the labkit module, as well as the bit-clock from the ac97 module that interfaces with the audio codec. In addition, the two crucial input signals are the “ready” signal and the 16-bit “right_in_data”. “Ready” signals that audio data is valid, and the most significant 16 of 20 bits in the audio data “right_in_data” are stored into the RAM_dp whenever the RAM_controller detects a rising or falling edge in the “ready” signal. The RAM_dp stores 2048 lines of data, and is addressed with 11 bits. In general, the xfft1024 module will process 1024 lines of data, so it will index through half of the data in the RAM_dp, while the newly inputted audio data is stored in the other half. Since the rate of FFT computation is much higher than the rate of input data storage, the xfft1024 module must wait until a rising or falling edge transition of the most significant bit of the waddress indicating that 1024 lines of data have been stored, before it can begin computation again. The following Figure 3 shows a state transition diagram of the FFT_controller module that sends control signals to the xfft1024 module.

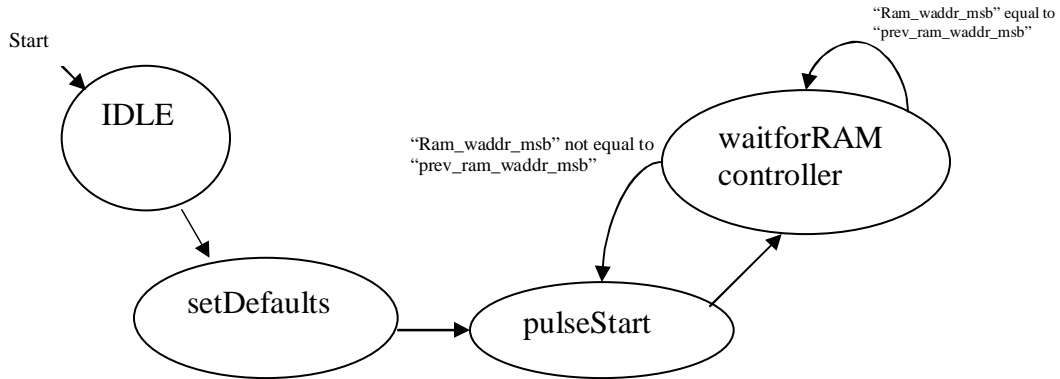


Figure 3: State Transition Diagram of FFT controller

Upon a global reset signal, the FFT controller enters into the IDLE state. On the next clock cycle, it enters the setDefaults state where parameters are sent to the Xilinx xfft1024 module telling it the size of the transform to be computed (nfft), a scaling schedule (scale_sch), whether to perform a forward or inverse transform (fwd_inv), etc. These parameters are registered into the xfft1024 module by the FFT controller's pulsing of the respective write enable signals. Next, the FFT enters the pulseStart state, where it commands the xfft1024 to begin computing the transform. The FFT controller waits for the RAM controller to finish writing 1024 data lines to the RAM_dp in the waitforRAMcontroller state before returning to the pulseStart state, and repeating the transform operation.

After the xfft1024 module computes the 1024-point transform of the input data, the respective imaginary and real frequency output points (xk_im) and (xk_re) that are indexed by xk_index are sent to a multiply/accumulate unit. In the multiply accumulate unit, the squared magnitude of each point in the transform is computed. Next, xk_mag_sqd for each point along with the corresponding index that ranges from 0 to 1023 (rd_addr) is sent to the comparator unit. The comparator determines the peak of the frequency spectrum, and sends the index of the peak to a lookup table called notesROM. The lookup table matches the index of the peak (max_index) to a corresponding 5-bit note value representing the note to be drawn. In this project, the output notes are limited to one octave between middle C and high G on the treble clef, and all other note frequencies are considered out of range and are displayed with either a quarter rest or an empty bar respectively depending on whether the out of range note is too low or too high.

2.2 Video Display (Shi Ling Seow)

The video component is implemented on a VGA display with a resolution of 1024 x 768 and a screen refresh rate of 61.74 Hz. The timing values are listed in Table 1 below. This component consists of three modules as shown in Figure 4: the BRAIN module, WRITE¹ module and the VIDEO DAC. The CORE Generator™ tool is used to generate a built-in RAM that can store 786432 one-bit data and a ROM that can store 45056 one-bit data. The display is divided into 384 32 x 64 blocks where each block is an image of a single note as shown in Figure 6.

Table 1: Timing values for video display

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
1024x768, 61.74Hz	64.8	1024	16	96	176	768	1	3	28

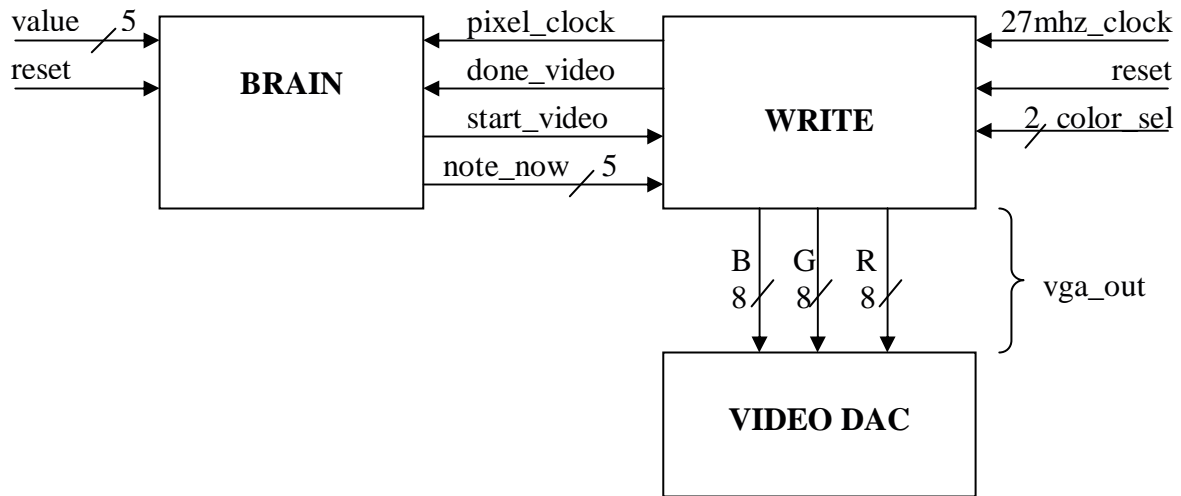


Figure 4: Block diagram of the video component

BRAIN module

Since the output rate of the pitch detector is only approximately 3 Hz while the video has a much higher screen refresh rate (61.74 Hz), a separate module is needed to regulate the data flow between these two components to prevent the video from displaying the same note consecutively and filling up the screen too fast. The BRAIN module runs on a 64.8 MHz clock and takes in a *reset* signal, a 5-bit data (*note_in*) which indicates the note that has been detected by the pitch detection component, and a *done* signal from the WRITE module. Then BRAIN outputs a 5-bit data (*note_now*) which selects the note to be displayed and a *start* signal to indicate that the WRITE module should display the new note on the screen. BRAIN has four states as shown in Figure 5. When the *reset* button is pressed, the state will return to IDLE and the *start* signal is reset to zero. During the IDLE and RESTART state, the data that is received from the pitch detector, *note_in*, is stored in a variable called *note_temp*. *Note_now* is set to be the same as *note_temp* for all the states. The DECIDE state will loop back to itself until the pitch detector outputs a new note that is different from the previous one. Then the *start* signal is set high and the state transitions to WAIT where *start* is kept high until a *done* signal is received from the WRITE module. Then the state transitions back to RESTART and the steps are repeated.

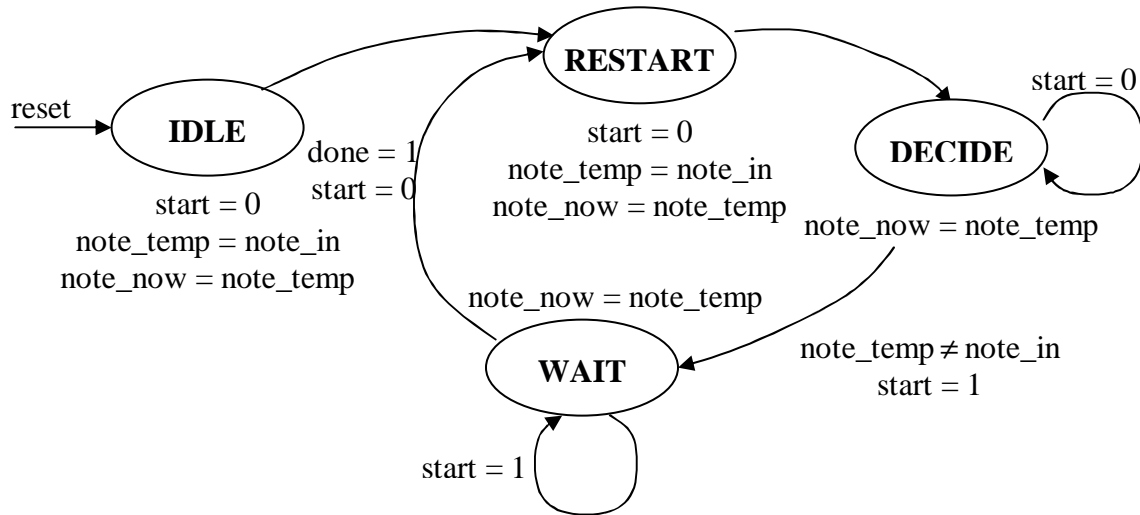


Figure 5: State transition diagram of the BRAIN module

WRITE module

The WRITE module consists of three main sections: DCM module, internal signals generator, and a finite state machine. The DCM module generates a 64.8 MHz *pixel_clock* using a 27 MHz clock. The internal signal generator takes in a 64.8 MHz clock and generates a *hsync* and a *vsync* signal for the video display timing. The finite state machine consists of a read section and a write section. There is also a section which initializes the RAM with data from the ROM to display an empty bar when the *reset* signal is pressed. This method of initializing the RAM is used because initializing from a .coe file would take too long to simulate due to the size of the RAM. The RAM is used to keep a one-bit data for all the pixels on the display, with a zero to represent the background and a one to represent the foreground. The ROM stores 22 32 x 64 bitmaps of the displayed notes, each bit representing a pixel on the screen as shown in Figure 6.

This design is simple and does not require too many pointers and counters as the data for the video display is obtained directly from the RAM. No doubt this would require the RAM to be extremely large to be able to store all the pixels but since our display only shows two colors, the RAM only has to store one-bit data for each pixel. The *write_count* variable keeps track of the next writing point on the RAM and the *read_count* variable keeps track of the next reading point on the RAM. *Pointer* points to the next block of data that needs to be written. *Rom_count* is used to point to the ROM address where the data to be written into RAM is obtained from.

When the *reset* signal is high, the state transitions to IDLE. Then the SET state loops back to itself until two cycles before the blanking period begins. The ROM address is first set to point to the block where the image of an empty bar is stored before the state is transitioned to SETRAM where the ROM address is incremented and *write_count* is set to the *pointer* value. Then both the ROM address and the RAM address are incremented and the write enable signal is set to high in state SETROM. The WRITE state then loops back to itself and continues to increment both the ROM and the RAM addresses until it

reaches the end of the RAM buffer. Now the write enable signal is set back to low and *write_count* is reset back to address zero. The WAIT state loops until one cycle before the blanking period is over. Then it transitions to state READ the RAM address is incremented and the data from RAM is read at every *pixel_clock* cycle until the one cycle before the horizontal blank period. READ then transitions to READWAIT which loops until one cycle before the end of the horizontal blank period. Then the state transitions back to the READ state to continue displaying on the screen. This back and forth transitioning between READ and READWAIT state continues until the end of the RAM address. If the *start* signal from the BRAIN module is low, READWAIT transitions to WAIT. But if the *start* signal is high and if the *pointer* is pointing at the first block right after the edge of the screen, (see Figure 6) READWAIT transitions to TREBLERAM where the ROM address is incremented and *write_count* is set to the *pointer* value. The ROM at this point is pointing to the bitmap of a treble clef so that a new treble clef would be written at the beginning of the bar. Then both the ROM address and the RAM address are incremented and the write enable signal is set to high in state TREBLEROM. The TREBLEWRITE state then loops back to itself and continues to increment both the ROM and the RAM addresses until a block of data has been written into RAM. Now the write enable signal is set back to low and *write_count* is set to *pointer* which is pointing to the next block where the next note will be written. Then the state transitions to WRITERAM, WRITEROM, and WRITING where all the procedures in TREBLERAM, TREBLEROM, and TREBLEWRITE are repeated but with the ROM address set to the *note_now* value that was outputted by the BRAIN module. If the *pointer* is not pointing at the first block right after the edge of the screen and the *start* signal is high, READWAIT transitions to WRITERAM instead and skips the step of writing the treble clef. At the end of the WRITING state, the *done* signal is set to high and the state transitions to WAIT. Figure 7 shows the state transition diagram for the WRITE module.

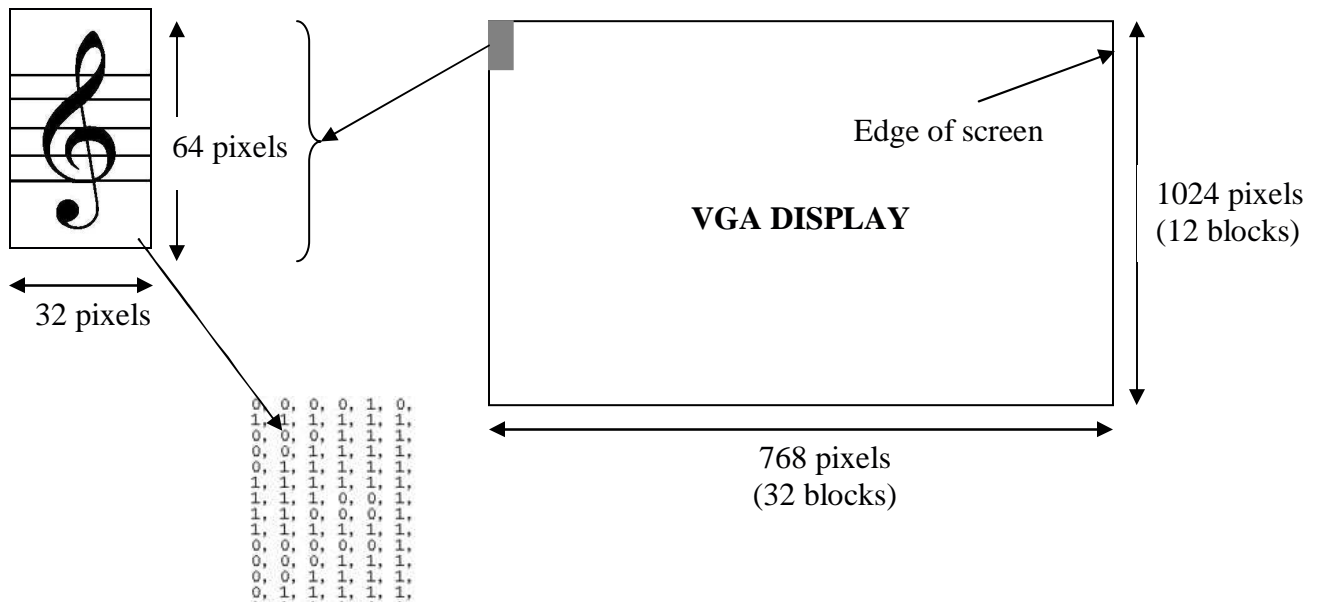


Figure 6: The VGA display is divided into 32 x 12 blocks. Each block contains a bitmap of size 32 x 64 pixels which is stored in the RAM.

The sampling rate turned out to be a crucial decision of the pitch detecting module. Although the input audio data entered at a frequency of 48kHz, I decided to decrease the input RAM storage rate by a factor of 16 to 3kHz, so that the frequency spectrum produced by the FFT module would have a higher resolution. With a sampling rate of 3kHz, the 1024 point transform allowed each bin to have roughly a 3-Hz resolution. Since the frequencies corresponding to the notes on the treble clef display ranged from 261 Hz (Middle C) to 784 Hz (High G), the sampling rate of 3kHz allowed approximately 20% of the points in the transform (between point 86 and point 278) to be matched to notes. If the frequency were determined to be out of range, meaning that the maximum point was below point 86 or above point 278, then the pitch detector would instruct the video display to output a rest or an empty bar respectively.

The relatively slow sampling rate allowed for higher probability of hitting a note value when singing, but compromised the number of notes that could be captured. We discovered that when an input midi file contained notes that changed too quickly, the pitch detector could only produce one of every few notes.

3.2 Video Display (Shi Ling Seow)

There are various ways to implement a video component and each has its own strengths and weaknesses. The main objective of my design is to minimize the number of pointers used to reduce complexity. The design is relatively simple and straightforward as it does not require too many pointers and counters since the data for the video display is obtained directly from the RAM. No doubt this design would require a large RAM to store data for all the pixels but since our display only shows two colors, the RAM only has to store a one-bit data for each pixel.

I also chose to not initialize the RAM with a .coe file because simulation would take too long due to the size of the RAM. Therefore, I wrote a simple three state FSM to initialize the RAM whenever the system is reset. The video display will miss the first read cycle but since the refresh rate is so high, the human eye will not be able to detect this latency.

Since the output rate of the pitch detector is only approximately 3 Hz while the video has a much higher screen refresh rate (61.74 Hz), a separate module is needed to regulate the data flow between these two components to prevent the video from displaying the same note consecutively and filling up the screen too fast.

All the reads and writes are implemented on a single FSM instead of using a major and minor FSM. I chose to do it this way because the video already has very complicated timing constraints and splitting the controls into several modules would just add more timing and synchronization issues to the video component.

4. Testing and Debugging

4.1 Pitch Detector (Yun Wu)

Debugging the pitch detector was extremely cumbersome mostly because of the xfft1024 module. As it turned out, there was not an easy way to simulate the Corgen FFT block,

so most of the debugging was done by sending signals out to a logic analyzer. Everything after the FFT module as well as the `fft_controller` and `ram_controller` were simulated using Max+plusII and ModelSim. The following Figures 8, 9, and 10 show the `ram_controller`, `fft_controller`, and everything after the FFT module in simulation.

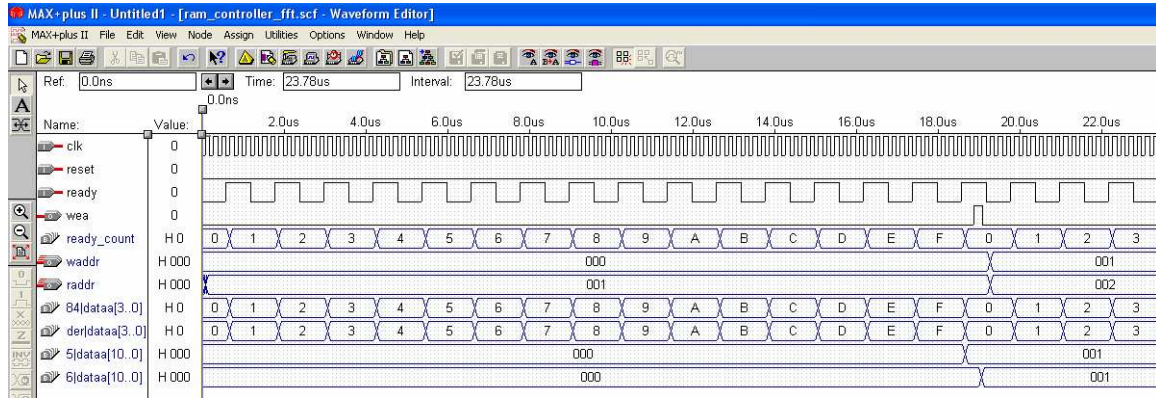


Figure 8: Ram_controller in simulation

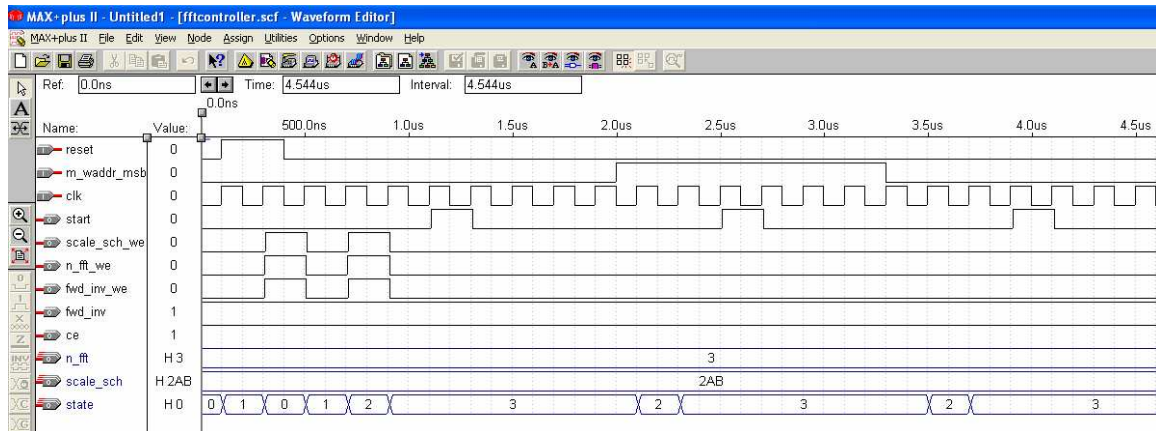


Figure 9: FFT_controller in simulation

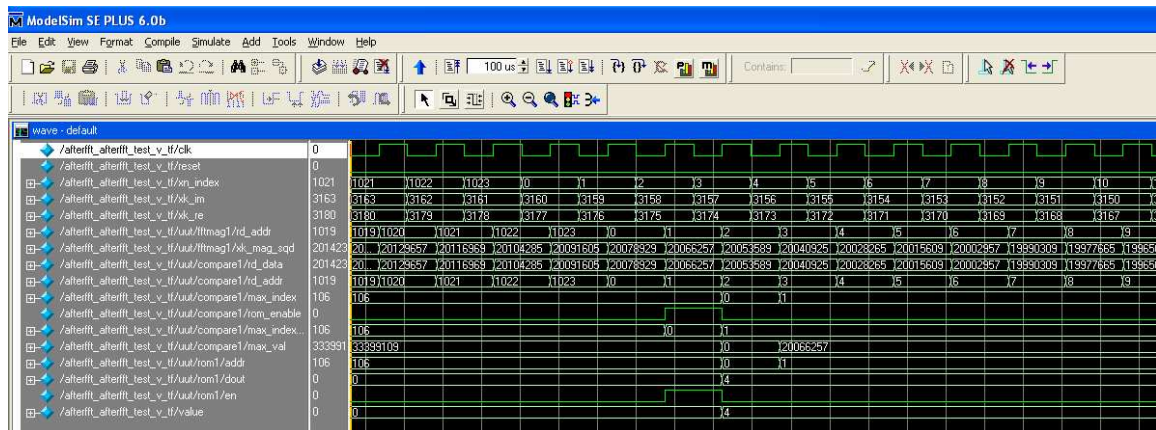


Figure 10: Everything after the FFT module in simulation.

4.2 Video Display (Shi Ling Seow)

I coded and tested my design one layer at a time as the video component has very complicated timing issues. By doing so, I could detect errors easily and mistakes can be detected earlier before the code becomes too complex. First, I tried displaying data from a RAM by just reading directly from RAM without any write procedures. The next step was harder as I tried to write from ROM to RAM and then read from the RAM. At this point I wasn't writing in blocks and was merely writing sequentially to familiarize myself with the READ and WRITE timings. Once I have managed to do that, I went on to write in blocks. This part took up the bulk of my time as it was difficult to keep track of so many pointers in my code. But after that was done, polishing the details was relatively simple.

For the purpose of testing and debugging, besides displaying the data on the video screen, I also simulated all my codes using Max+PlusII for simple codes that did not require the use of RAM and ROM such as the BRAIN module and used ModelSim for simulating other modules as it ModelSim is linked directly from Xilinx. The DCM module could not be simulated in ModelSim so a different clock was generated for the purpose of simulation. Also, because the timing specification of the video is very long, certain timing constants were reduced for faster simulation. Figure 11 shows the simulation for the write timing and Figure 12 shows the simulation for the read timing. Figure 13 shows the simulation for the BRAIN .

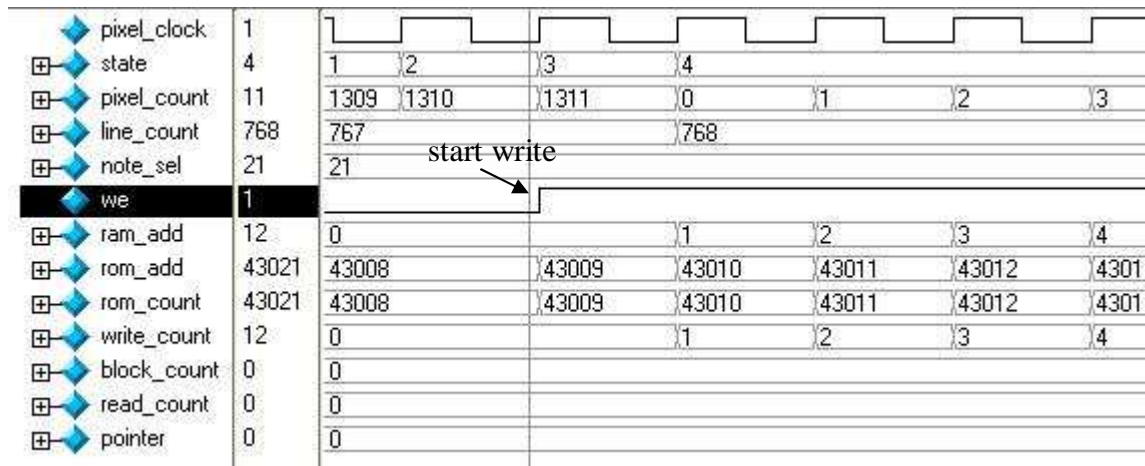


Figure 11: Simulation of write timing for video. The ROM address is provided one cycle before write enable is set high so that the data from that ROM would be available on the next cycle to be written into RAM address zero.

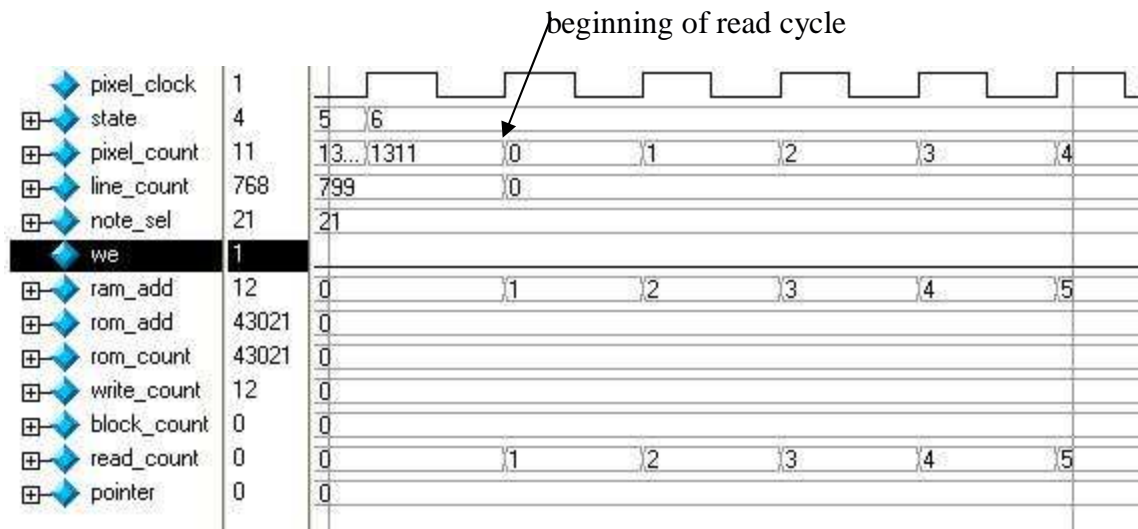


Figure 12: Simulation of read timing for video. The first RAM address to be read is provided one cycle before the beginning of the read cycle so that the data from RAM would be available when the read cycle begins.

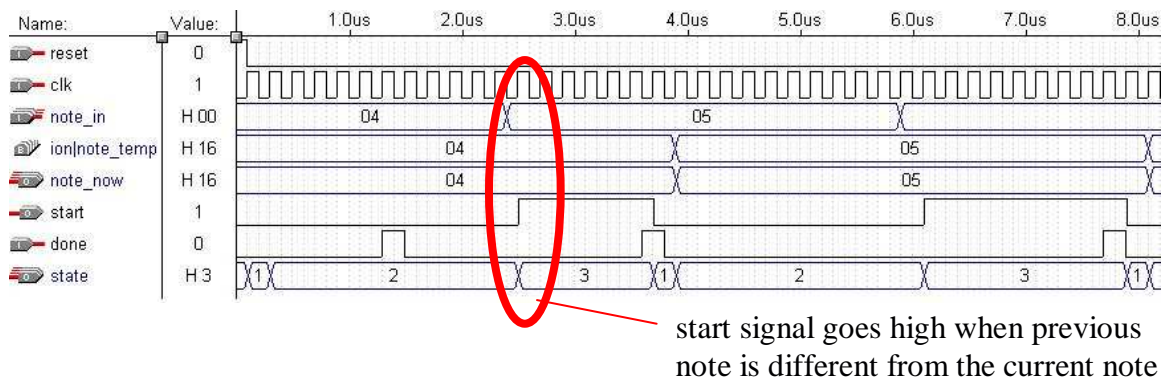


Figure 13: Simulation of the BRAIN module

5. Conclusion

Our system demonstrates that input audio data can be processed by an FFT module, and the corresponding note can be determined by detecting the peak of the input frequency spectrum. The notes can be directly fed into the video display module, which outputs the notes one at a time onto a staff. The system is reliable for determining and displaying the frequency of preset sinusoidal signals by a signal generator, as well as determining the corresponding note values to be displayed. Our system is mainly limited by its input sampling rate of 3kHz, as we sometimes missed a few notes when the music changed quickly or when the notes included harmonics out of our range. Future additions could include a larger note range, as well as detecting the duration of notes, displaying different types of notes depending on duration, scrolling screen and filtering input from the microphone so that pitch detection can be performed from a note sung into the microphone.

Acknowledgements

The authors would like to express their sincere gratitude and appreciation to several people, without whom this project would not be possible. First we would like to thank Nathan Ickes for the wonderful new lab kit and all of the technical expertise he provided. Next, we would like to thank Keith Kowal for his video expertise, as well as the rest of the friendly lab support staff for lending us microphones, speakers, and various peripherals. Finally, we would like to thank our TAs: Charlie Kehoe, Jenny Lee, and Chris Forker for countless hours of patience, support, encouragement and help, and Prof. Anantha Chandrakasan for his kindness, time and effort in making 6.111 a truly memorable class.

References

- [1] “Fast Fourier Transform v3.0” Product Specification. Xilinx Logicare. 5/21/2004.
- [2] Ickes, Nathan “Audio Input and Output”
<<http://www-mtl.mit.edu/Courses/6.111/labkit/audio.shtml>>
- [3] Ickes, Nathan. “Methods for Programming the Labkit”
<<http://www-mtl.mit.edu/Courses/6.111/labkit/configuration.shtml>>
- [4] “LM4550 AC '97 Rev 2.1 Multi-Channel Audio Codec with Stereo Headphone Amplifier, Sample Rate Conversion and National 3D Sound” National Semiconductor. 8/2003.
- [5] Chuan, Ching-Hua and Zhu, Kevin “Guitar Scores Interpretation: Transforming Audio into Guitar Tab Scores” < <http://www-scf.usc.edu/~chinghuc/ise599.html> >