Voice Training Karaoke Machine

6.111 Final Project

Masood Qazi
Zhongying Zhou

6.111 Spring 2006
May 17, 2006

Abstract:

A Karaoke has been implemented for the purpose of vocal training. It will have as its two primary inputs the user's singing and the notes from the sheet music of the vocals for a selected song. The user's voice will be recorded through a microphone interfaced to an analog to digital converter. The power spectrum of the user's voice will be computed and analyzed to infer the note being sung. The system will then compare the user's singing, by pitch and rhythm, with what is described in the sheet music. This comparative analysis is presented in a meaningful way to the user through a VGA display. Finally, the user will also have access to audio outputs such as the sequence of correct tones and the user's recorded voice through headphones. This system demonstrates important design concepts such as modularity and integration. In particular, testing was made more efficient by debugging smaller chunks such as the Fourier Transform or the Video Display at one time. A secondary round of testing was needed once the blocks were integrated.

Table of Contents:

List of Figures:

List of Tables:

1 Overview

Many times, people cannot distinguish whether they are singing the correct pitch in a song. This is may be due to the inability to hear their own voice, or due to the presence of background instrumentals of a song. As a result, many people might consider themselves to be good singers and embarrass themselves if in reality as they sing off-key. The purpose of this project is to allow users to have a more objective feedback to their singing. There are two songs stored in the memory and the playback tempo can be modified. The user can also pause, fast forward or rewind through songs. The input singing will be analyzed in the frequency domain to select the pitch they are singing. The system will then display the waveform and pitch that the singer is singing compared to what is described in the sheet music. Additionally, the frequency domain analysis will also be displayed to provide additional sources of feedback. Finally, the user will also have access to audio outputs such as the sequence of correct tones and the user's recorded voice through headphones.

Shown in Figure 1 is the top-level block diagram that gives the overview of the system. There are four major divisions in the system:

1. Interface for the audio input through microphone with A/D converter.
2. Music reading component that controls which note of the song is outputted.
3. Analysis of the spectrum of the user's audio input to decide the most appropriate note.
4. Comparison of information by producing a visual of spectrum analysis and input notes.

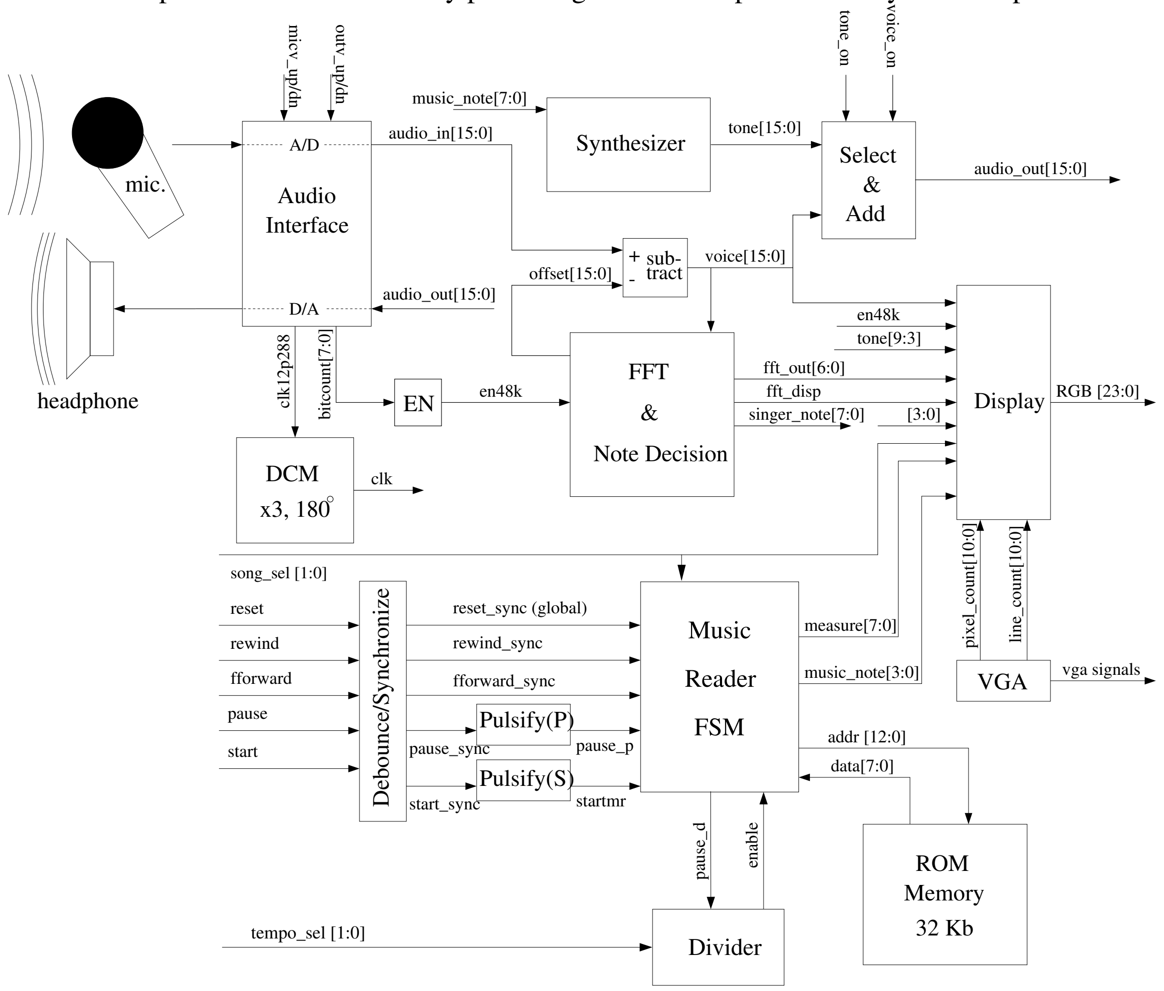

Figure 1: Top-level block diagram

1.1 Operation

Music Reader

There are two songs stored in memory: "Crazy" by Britney Spears and "She Bangs" by Ricky Martin corresponding to the song select value of 0 and 1 respectively. The user can pause, rewind or fast forward through the song. Also, the user can adjust the tempo of the song once the system is reset. There will be a 2 bit tempo select switch [1:0] corresponding to largo, andante, moderato, and allegro. (Table 1 : Note duration and switch values for different tempos) Note that the values are calculated based on the equation that the duration for a quarter note is four times that of a sixteenth note. The music reader will use this information accordingly.

Table 1 : Note duration and switch values for different tempos

Tempo	Beats/Minute	Switch [1:0]	16th Note Duration (ms)
Largo	60	00	250
Andante	90	01	166.7
Moderato	120	10	125
Allegro	150	11	100

To ensure proper operation of the music reader, the lower 7 bits of the output address is wired to the lower 7 bits of the LED display. The highest bit of the LED toggles between on and off every time a sixteenth note has passed. This displays the current rhythm. (Figure 2 : LED display for address and rhy)

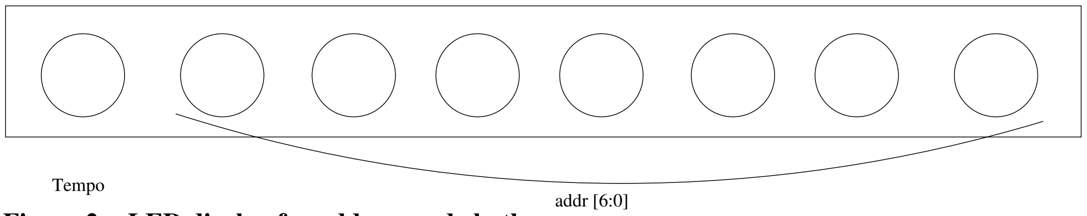

Figure 2 : LED display for address and rhythm

Display

The display module gives a visual cue to the singer on his/her performance. The synthesized square wave is displayed at the top right corner and the singer's input waveform is displayed below that. Each waveform consists of 400 7-bit data and a maximum amplitude of 128. At the bottom of the screen is the frequency domain spectrum. However for the frequency display, each of the data appears as a filled box that is 4 pixels wide. (Figure 3) In addition, the user can freeze the graphs to observe the waveforms at an instant in time. One final feature of the display is on the top left half of the screen, the current song title and artist are displayed as well as the lyrics corresponding to the measure of the song that the system is playing.

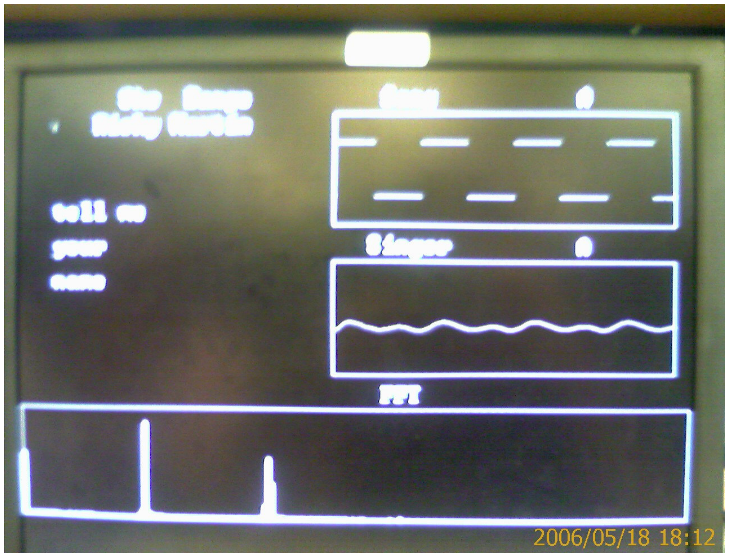

Figure 3: Positioning of outputs on visual display

Testmode

The Karaoke playback of the song can be paused at anytime by entering a testmode which is indicated by bringing switch[3] high. In this testmode, the fast forward and rewind buttons change function to incrementing and decrementing the synthesizer note. Basically, the system turns into a voice tuning machine in which the user can tune his or her voice to a fixed note while observing the audio wave form and the power spectrum in real time. Upon exiting the testmode, the song resumes from where it left off.

1.2 Implementation

Music Reader

The inputs for pause, play, rewind, and fast forward are implemented as buttons. The data for the tune of the song is stored in a block ROM that is 4096 x 8 bits. For one block of data (8 bits), the upper 4 bits represents the octave of the note in binary and the lower 4 bits is reserved for the actual note. In addition, the lower 4 bits also encodes for rest, begin song and end song signals. (Table 2: Encoding of lower 4 bits of a note in Hex) Choosing this design scheme allows each note to have a constant duration of one sixteenth note. An FSM module will be built to control which address of the memory the system is reading from.

Table 2: Encoding of lower 4 bits of a note in Hex

note[3:0]	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Encodes	A	A#	B	C	C#	D	D#	E	F	F#	G	G#	rest	beg	end	--

Display

There are two major components of the visual display: waveforms and alphanumeric characters. The following blocks are all instantiated within the Display/Comparator block. To display the waveforms, a display graph module is used. A modified version is used for the FFT module. Finally, the alphanumeric display is processed within the Compare/Display block. It uses an existing code that outputs the display when given a string of 8 characters and position desired. All of the resulting RGB signals are then OR'ed to produce the final output.

The remainder of this document will describe the Voice Training Karaoke Machine's design and performance specifications followed by the testing and debugging procedures used in its development. Finally, the shortcomings of the current design and possible improvements will be explored.

2 Audio Interfacing, Spectrum Analysis, Note Synthesis (MQ)

2.1 Representation of Notes (MQ)

A note sung by a person generates a periodic waveform whose frequency corresponds to the pitch. This period corresponds to the fundamental frequency: the lowest frequency component in the Fourier transform representation of the audio signal; however, the quality and uniqueness of a person's voice depends on contributions from higher harmonics of the fundamental frequency in both magnitude and phase relative to the fundamental.

Classical music theory maps notes to the frequency domain in a geometric spacing given by:

$$f_n = f_{A4} \bullet 2^{n/12}$$

This formula indexes a note by an integer n that represents its distance in "half steps" from the note A4 (the note A in the fourth octave where f_{A4} = 440kHz). Multiplying or dividing a note's (fundamental) frequency by a power of two does not change the note, but only shifts the octave. Thus a note can be indexed by a set of twelve integers m from 0 to 11. Shown in Table 3 is an example of frequencies, and indices for the notes in octave 5. Shown in Figure 4 is a graphical representation of the cyclical indexing of notes with the clockwise direction representing increasing frequency.

Table 3: Information related to music notes (example for octave 5)

Note:	f [Hz]	n	m = n mod 12	FFT "k" range	Half period in clock cycles
A	880	12	0	73-77	20945
A#	932	13	1	78-81	19770
B	988	14	2	82-86	18660
C	1047	15	3	87-91	17613
C#	1109	16	4	92-97	16624
D	1175	17	5	98-103	15691
D#	1245	18	6	104-109	14811
E	1319	19	7	110-115	13979
F	1397	20	8	116-122	13195
F#	1480	21	9	123-129	12454
G	1568	22	10	130-137	11755
G#	1661	23	11	138-145	11095

Thus, the goal of my (Masood Qazi) component of the project is to obtain a digital representation of the singer's audio, examine its spectral content, and infer what note is being sung. In addition, the correct tone will be synthesized as audio feedback for the user along with his or her own voice. I will design for the range of human vocal frequencies which extends from roughly 80Hz to 1.2 kHz.[1]

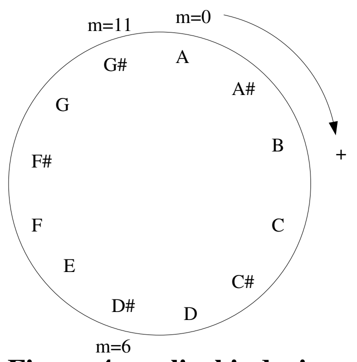

Figure 4: cyclical indexing of notes

2.2 AC97 Audio Interface (MQ)

[1] http://www.tnt-audio.com/topics/frequency_e.html

Shown in Figure 5 is a block diagram for the AC97 Audio interface. At the heart of the system is the LM4500 chip that performs analog to digital and digital to analog conversion at a rate of 48kHz (i.e. 48,000 discrete time samples per second). In order to give the LM4500 commands and read its data, serial to parallel conversion must be performed since only two wires are used for input and output. Illustrated in Figure 6 is an example of serial to parallel conversion of data. For a given range of clock cycles—indexed by the bitcount variable—the current value on the serial data port is interpreted as one of the bits of a multi-bit signal. After all the bits have been red sequentially, they are simultaneously presented on a bus of multiple wires. As a result, the entire converted signal is available to the rest of the system in one clock cycle.

All the tasks in the AC97 interface are referenced to a common time axis through the bitcount variable which cyclically counts from 0 through 255. These 266 clock cycles correspond to one "frame" in which one new data sample for both left and right channels can be received and submitted. In addition to receiving and sending audio data, the interface has a command loop to cycle through a sequence of commands on a frame-by-frame basis. As a result the user can adjust the microphone and headphone volumes because these variables are internally updated in the LM4500 chip frequently when the corresponding command is entered in the command loop. A cycle through all the commands occurs on a millisecond time scale.

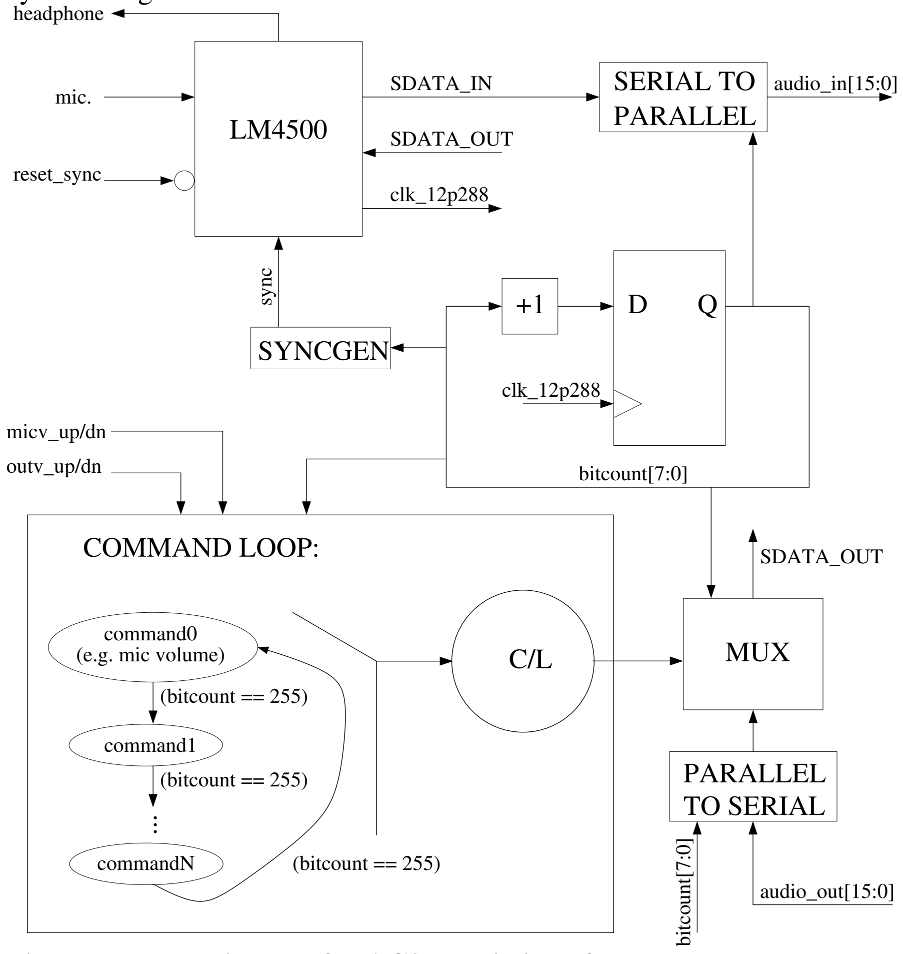

Figure 5: Block diagram for AC97 audio interface

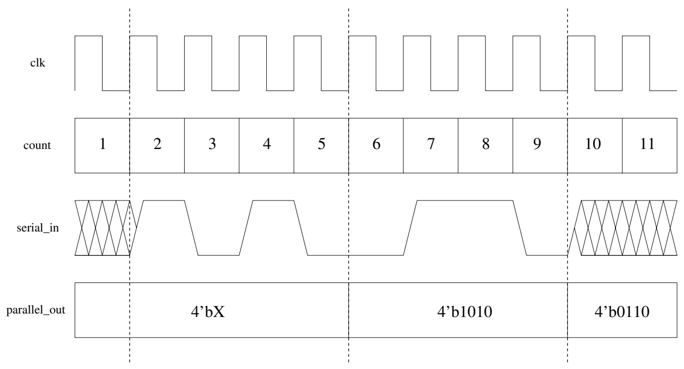

Figure 6: Example timing diagram illustrating serial to parallel conversion

2.3 Note synthesis

Shown in Figure 7 is the block diagram for the synthesizer. This module is best understood by tracing the data path through the block diagram. The specified "music_note" is mapped to a half period in terms of number of system clock cycles (which operates at a rate of 36.864MHz). This implementation is restricted to a discrete set of frequencies, but the very fast system clock relative to the audio sample rate allows for very high precision in note specification. An example of half period lengths can be found in Table 3for octave 5.

Next, a cyclic counter runs from 1 to the half period (returning to 1 thereafter). This counted value provides phase information for the periodic wave form to be synthesized. In the case of the simple square wave, this counting scheme was used to trigger the amplitude to reverse it's sign (in a 2's complement number representation) twice during one period.

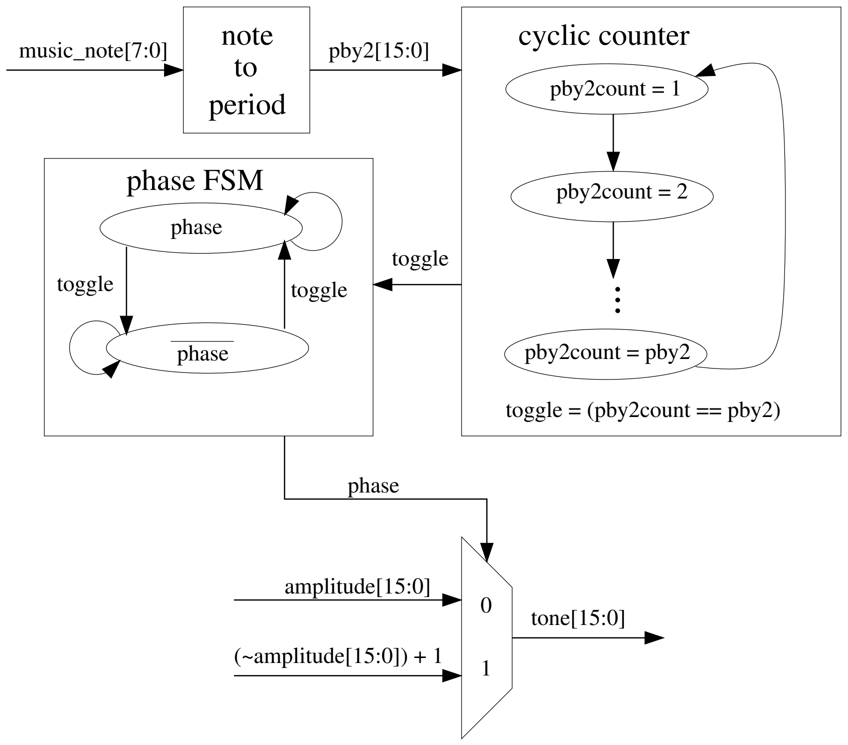

Figure 7: Synthesizer block diagram

2.4 FFT and Note detection (MQ)

Shown in Figure 9 is the block diagram for the FFT and note detection subsystem. Once, 4096 consecutive audio samples are written to a block RAM, the data is read in a serial fashion to an FFT module that will subsequently output evenly spaced discrete samples of the signal's Fourier transform. This behavior is illustrated in the state transition diagram of Figure 8. A length of 4096 corresponds to roughly 85ms which is roughly four times the period length of the lowest audio frequency we hope to observe. However, finite window length effects limit the ability to unambiguously resolve a note to the third octave. Below the third octave, notes are spaced too closely in frequency to be distinguished; only the octave can be extracted. However, real signals like human vocals contain a rich set of harmonics in addition to the fundamental frequency, thus the actual note (the k value in Figure 4) is inferred from a second, fourth, or eighth harmonic of the note. As a result, the karaoke machine successfully detected notes in all eight octaves.

Another issue to that compromised frequency resolution was a strong DC component in the signal due to built in offsets in the audio analog to digital converter. The strong DC component would suppress the dynamic range for non-zero frequency spectrum values due to finite precision arithmetic and it also introduced sinc-like ripples extending into higher parts of the frequency spectrum. This problem was solved by using the sample of the dc component in the spectrum to auto-zero the incoming data to the FFT module. This zero-mean data was also instrumental for the VGA display of the voice waveform as shown in the screenshot of Figure 3.

With this setup, the singer's note was determined from a simple maximum location algorithm described by the state transition diagram in Figure 10. The fact that the data was output in a serial fashion accommodated this approach very well. On the dot matrix display near the pushbuttons of the lab kit, the inferred note is displayed along with the octave number side by side with the correct note. Using this dot matrix display, the user can tune to a desired note *and* octave.

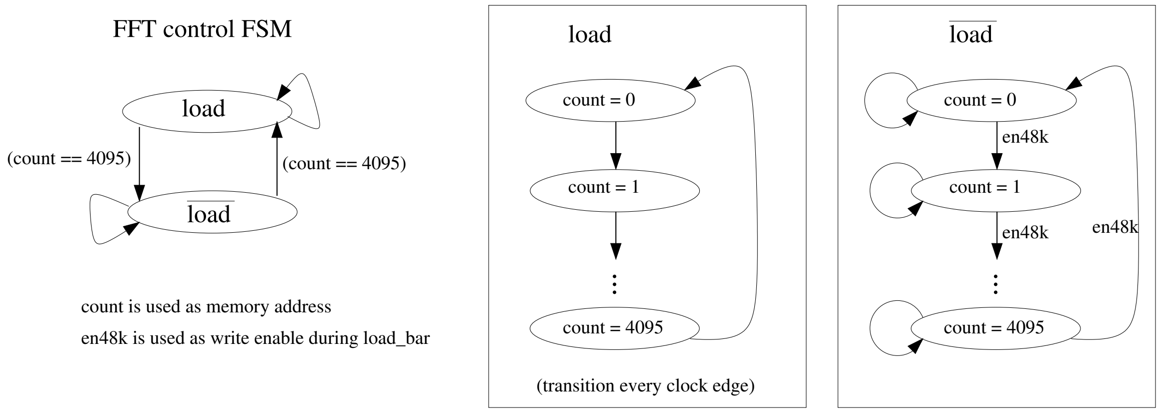

Figure 8: FFT conrol FSM

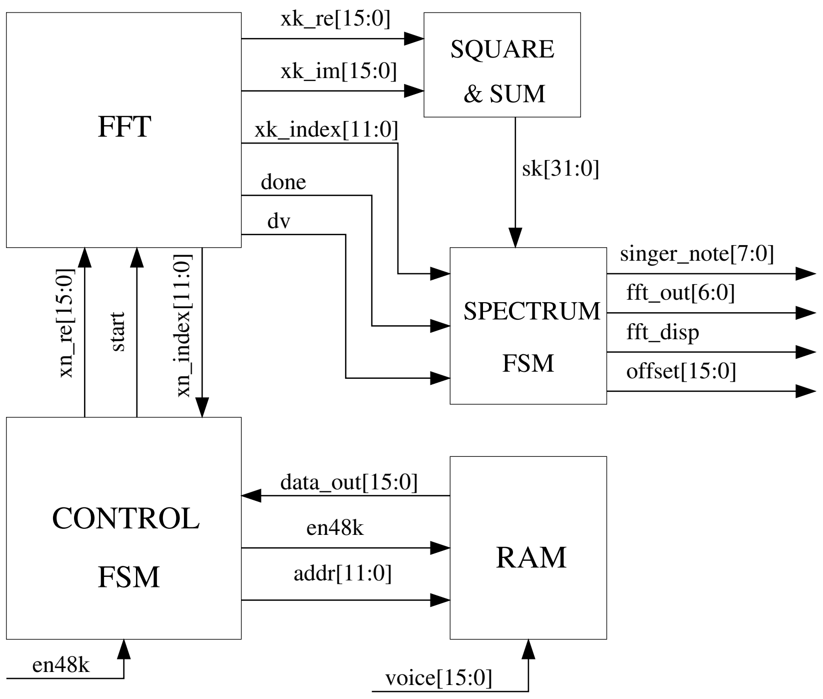

Figure 9: FFT Block diagram

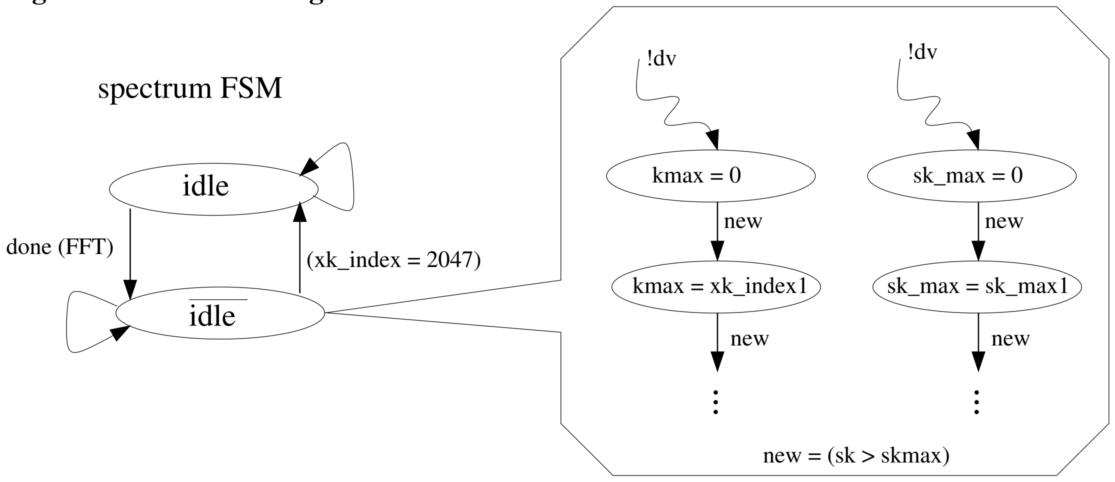

Figure 10: Spectrum processing FSM

3 Music reading and Comparison display (ZZ)

3.1 Pulsify

This module takes an input and converts the positive edge of the input signal into a pulse. It is triggered on the posedge of the global clock. The module will delay and invert the signal into another register. The actual output is the registered value of the actual signal ANDed with the delayed signal. Registering the output is important because it indicates to the music reader whether it needs to switch states.

3.2 Divider

This module takes a 2-bit tempo input and a pause_d input from the music reader. The tempo input selects the duration every time the user resets the system. This module counts using the 36.8 mHZ clock signals and outputs an enable pulse based on the tempo. (Table 4: Clock cycles corresponding to tempo) This enable signal activates the music reader FSM to update a new music note to the display and synthesizer modules and a new address to the block ROM. Also the music reader sends a pause_d input. If that signal is low, the divider will function as described above. However, if the signal is high, the divider will hold the current count value and only resume counting when pause_d is deactivated.

Table 4: Clock cycles corresponding to tempo

Tempo	# clock cycles
Largo	9216000
Andante	6144000
Moderato	4608000
Allegro	3686400

3.3 Music Reader Finite State Machine

The Music Reader takes the debounced rewind, fast forward signals and the pulsified startmr and pause_p user inputs. From the block ROM, it receives an 8-bit data corresponding to a note in the song. The outputs are the pause_d signal to the divider module, the 12 bit address counter to the block ROM and the lower four bits of the data, the music_note. The measure is the upper 8 bits of the address. This is because one measure is a whole note for the songs implemented so the lower 4 bits are omitted.

At reset, the state is initialized to IDLE and the address is offset based on the song_sel value. Also the output pause_d and music_note is zeroed. Otherwise, at every clock cycle, the state is updated and the pausing condition is checked. If the state is in PAUSE, then the input to the divider is high. When the divider sends an enable signal, the address is also updated. However, if the new address will be outside of memory, the old address is maintained. Music_note registers the current value of the lower 4 bits of data.

The FSM is implemented as a Moore machine and has a total of 7 states. At restart, the state is in IDLE and the increment is 0. The table shows the output signals corresponding to the states. (Table 5 : Outputs for FSM states) The state evolution is also outlined below. (Figure 11: State transition diagram for music reader)

Table 5 : Outputs for FSM states

State	Parameter	Increment	pause_d
IDLE	0	0	0
PLAY	1	1	0
PAUSE	2	0	1
REWIND	3	-4	0
FFORWARD	4	4	0
ENDSONG	5	0	0
BEGSONG	6	0	0

The BEGSONG and ENGSONG states exist since the user cannot fast forward or rewind past a song. However, the user can still rewind from the end of the song and continue normal operation from wherever the location the song rewound to. The case for the beginning of a song is similar.

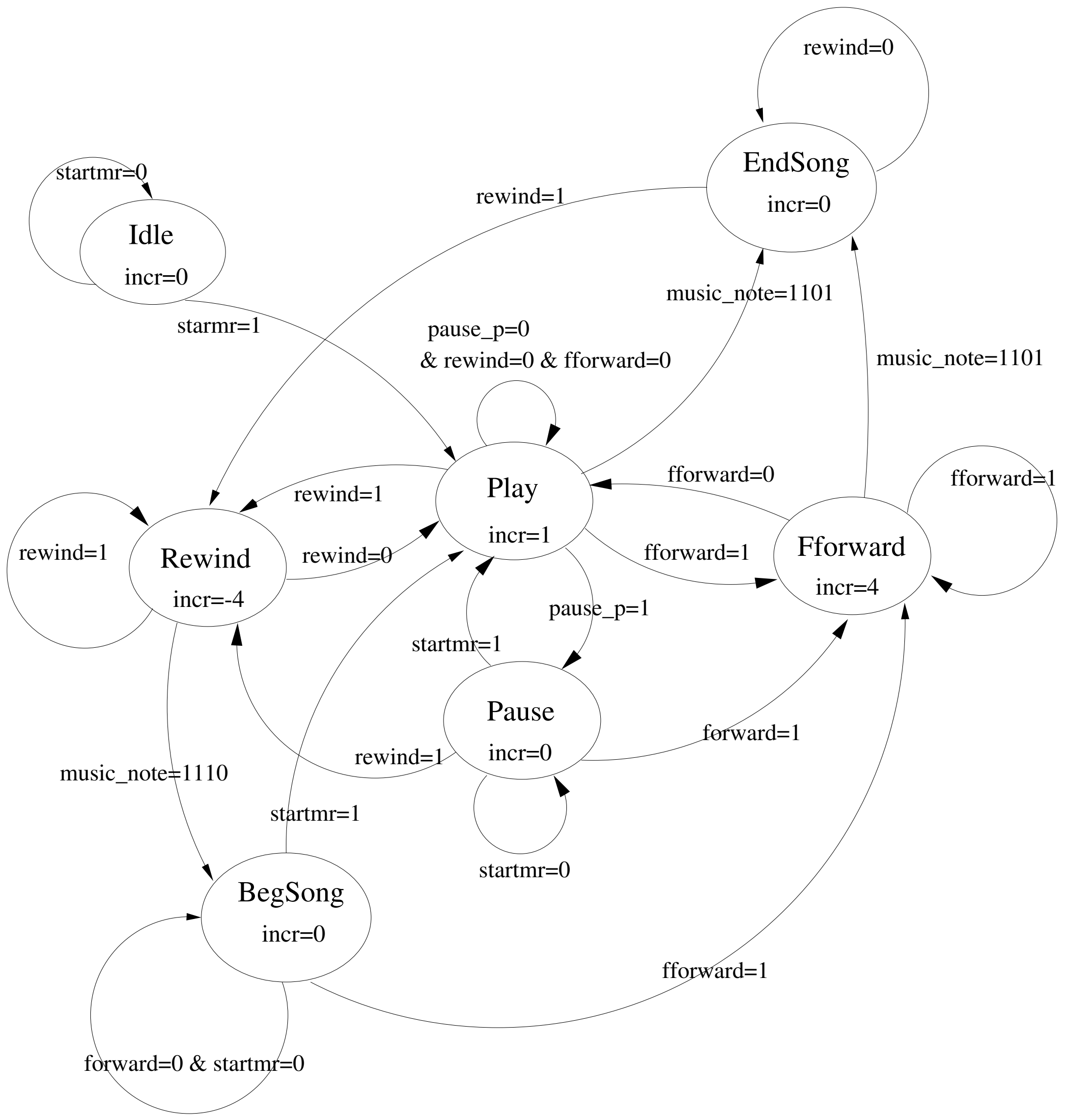

Figure 11: State transition diagram for music reader

3.4 Video Display

The display employs the 600x800 VGA timing scheme. This module instantiates all of the submodules described below and feeds them the correct inputs. The inputs into this module are pixel count, line count both 11 bits, music_note and singer_note, the copy_internal, and new_m, new_s, new_f which indicates new data, voice and fft_out inputs. Measure indicates which measure of the song is playing and song_sel indicates which song is playing. Graphs for music and singer are drawn using the appropriate update, copy and data signals. Also the fft data inputs are inputted to the draw fft module. Finally music_note, singer_note, measure, and song_sel choose which words are displayed. All of the RGB outputs from each component or ORed with each other to provide the final VGA outputs. (Figure 12: Display block diagram)

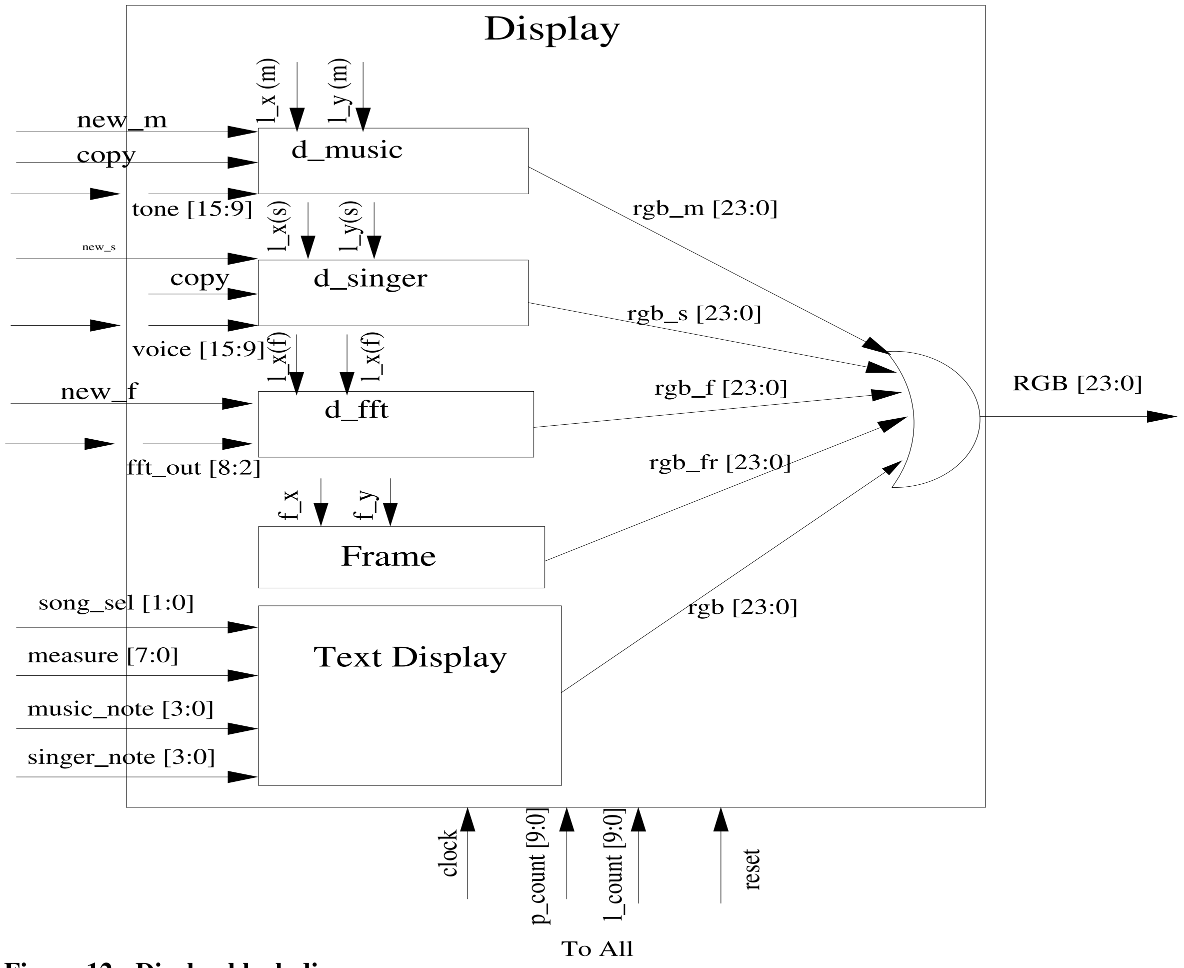

Figure 12: Display block diagram

Alphanumeric Display

This project also uses an alphanumeric display on the VGA. Existing code from the Fall 2005 6.111 class was consulted and used. To output the key that the singer and the song are producing, a case statement comparing the input with the encoding of a particular key is used. To display the song title and artist, a case statement uses the song_sel input to output the corresponding song. Finally, a similar case statement is used to output the lyrics based on the value of the measure input.

Frame

This module is a modified version of the prior rectangle module. Instead of drawing a filled rectangle, this draws the outer frame of the rectangle. The inputs are pixel count, line count, the top left coordinates of the frame, and the width and height of the frame which are 11 bits. The module outputs the corresponding RGB values. At every pixel and line count, the module first determines if the pixel count is at the left or right edge of the frame and within the height of the frame. If not, then it determines whether the line count is at the top or bottom edge and the pixel count is within the width. If either of the conditions is true, a white pixel is outputted (all 1s). Otherwise, the output is black (all 0s).

Draw Graph

Both the music and singer inputs are displayed as a waveform. This module takes pixel count, line count, and top left coordinates of the graph, which are all 11 bit inputs. The data input is 7 bits and is updated every 48 kHz. When the update signal is high the data is written into an array. When the copy_internal signal is high, it copies all of the 400 stored data into a backup array. (Figure 13: Draw Graph block diagram)

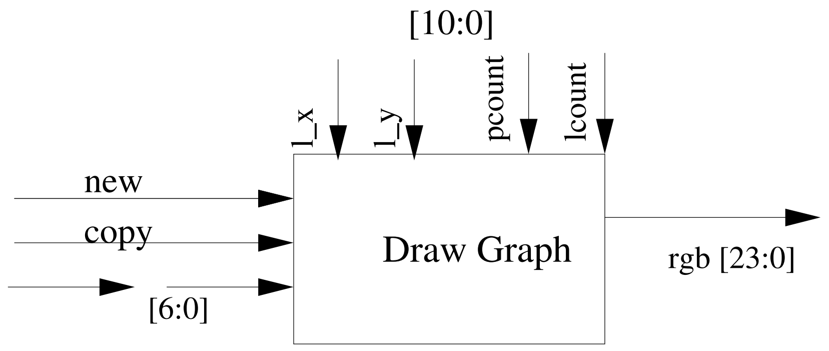

Figure 13: Draw Graph block diagram

There is a sequential block where the write pointer points to the position in the array that the next data will be written into. The read pointer points to the most recent data entry. If the update signal is high, the data is written and the write and read pointer is incremented. However if the copy internal signal is high, the entire array and the write and read pointer values are copied. This is used to latch the data once per frame so that the data will not change while the display is still drawing.

The combinational block determines the RGB values. It then uses the pixel count and matches it with the appropriate data value using the read pointer.

Draw FFT

The FFT module implements a similar idea. The module also takes pixel count, line count, and top left coordinates of the graph which are all 11 bit inputs. The data input is 7 bits and is written into the array when the update signal is high. The RGB outputs are then calculated. (Figure 14: Draw FFT block diagram)

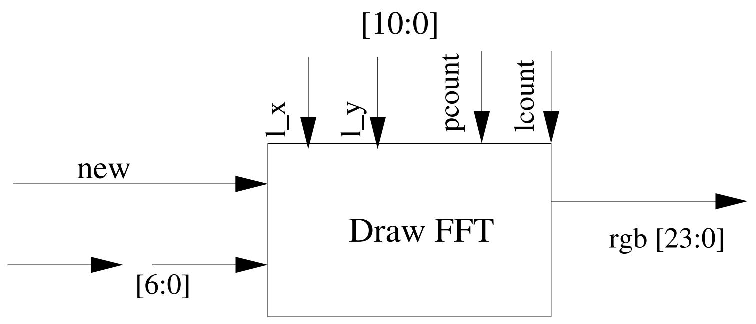

Figure 14: Draw FFT block diagram

The sequential block also updates the read and write pointer and writes the new data into the arrays. However, only 200 data samples are stored. The component that designates the RGB values is programmed in combinational logic. To allow four pixels per data, the relative index of the data is computed by shifting the output of pcount-l_x to the right by 2. Also, since the sequence of the FFT data is reversed as high frequency is most recent, the relative index is instead added to the write pointer. Finally, the output is white only if the pixel is below the data and above the bottom edge.

4 Testing and Debugging

There were many tests to ensure the proper function of these modules. For the music reading section, modelsim was used to ensure the proper operation of the divider, pulse and music reader FSM modules. The pulsify module was first tested because it was a simpler module. As shown below, pause_p is high for one cycle if the input transitions from low-> high. The delayed and inverted intermediate signal is displayed as well to ensure proper operation. (Figure 15: Pulsify module simulation waveform) The FSM caused some trouble because the address would not increment initially. The states were used as outputs to debug the FSM module and the error was found. The following is an example of operation of the FSM and a pause pulse input to the system. The state goes into the pause (2) and pause_d to the divider is lowered. (Figure 16: FSM simulation waveform demonstrating pause state) Afterwards, all of the modules were integrated and the address was outputted onto an LED display. After the address bits transitioned correctly, I added the data output from the ROM.

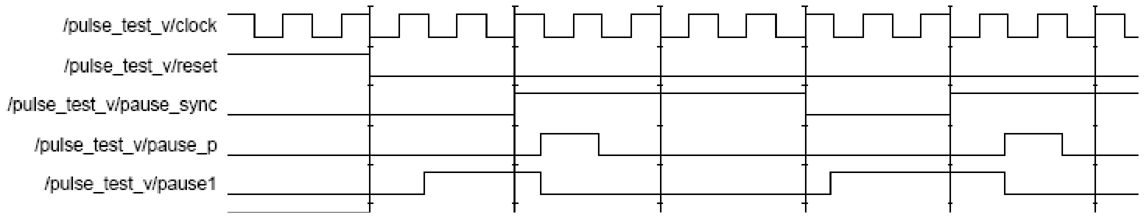

Figure 15: Pulsify module simulation waveform

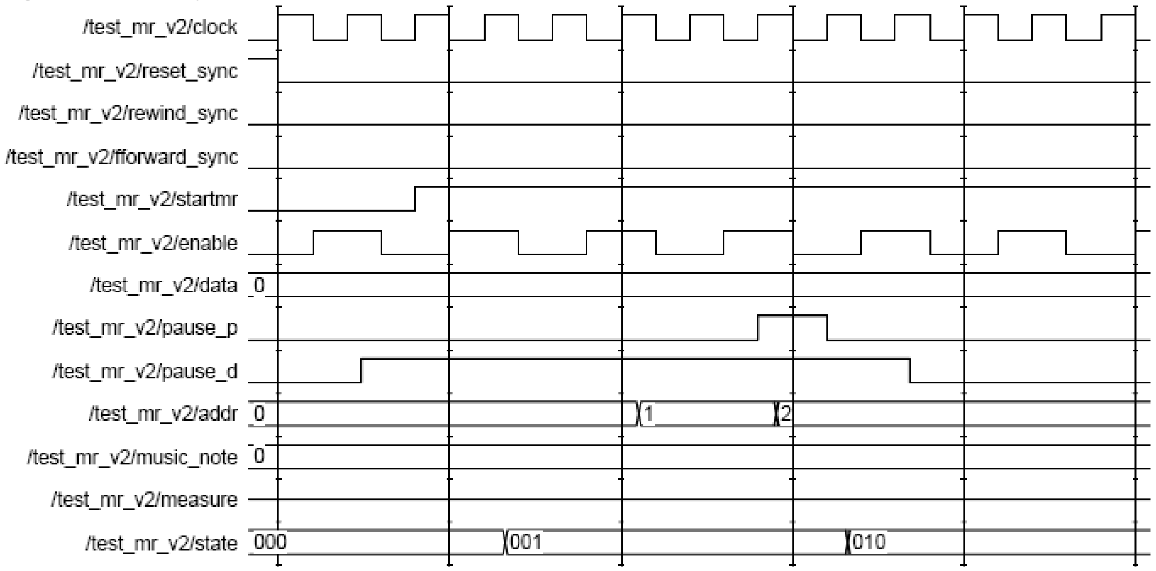

Figure 16: FSM simulation waveform demonstrating pause state

However the display module was tested visually with the VGA screen output. The first step was to build a module that stored input data into an array and displayed the array according to proper indexes into a line on the screen. However, the negative data would not display correctly so the decision was made so that all input data were positive numbers. Thus a buffer variable was added to convert the signed numbers into the unsigned numbers needed for the display. Once this was achieved, the FFT display was a modified version where the frequency axis was flipped due to the order of the input data. Additionally, each data corresponded to 4 pixels to have a clearer display.

The character display was also tested visually. The notes were compared to the ones displayed on the FPGA to ensure accuracy. The different songs were selected and the correct titles and artists displayed. In addition, when the song select switches were changed during singing, the title and artist still remained the same as expected. Finally the lyrics were tested by making sure they matched with the sheet music.

5 Conclusion

This project lends itself well to the real world. There may be many people who are unaware of how well they sing. With this new system, people will find objective ways to measure whether or not they are on tune with a song. Additionally, the visual displays allow the user to adjust their pitch and eventually perfect their singing talents.

One of the most challenging parts of this project was the design aspect. Because there were no basic block diagrams to follow, all aspects of digital design had to be considered. For example, since the A/D converter had an internal clock, to have the rest of the system compatible with that clock, the 36.864 MHz clock was used. However this may pose a slight problem for the VGA because the clock rate for 800x600 is 38 MHz. Once the clock rate was decided, the next challenge was how to approach the modules. Masood implemented a basic backbone structure for his components while Zhongying built the music reader first and then worked on the visual display. However, both approaches were successful because of the division of the modules.

All of the goals detailed in the proposal have been met. There were additional features such as the testing mode and the lyrics for the song that were added. However, if given more time, the system could still be improved. For example, a score could be displayed at the end of a song to reflect the singer's ability to stay on pitch.

One of the greatest lessons learned from the final project was the importance of modular designs. The visual display could be implemented without actual data; so long as the input data satisfied the requirements for that block. Each of the main blocks could function and be tested independently allowing a great reduction in debugging the modules. Overall this project gave insight into designing a complex system which could be divided and tested separately. As a result, the integration portion was successful due to well-defined input and output constraints.