Voice Recognition System

Jaime Diaz and Raiza Muñiz

6.111 Final Project

May, 2007

Abstract

This project attempted to design and implement a voice recognition system that would identify different users based on previously stored voice samples. Each user inputs audio samples with a keyword of his or her choice. This input was gathered but successful processing to extract meaningful spectral coefficients was not achieved. These coefficients were to be stored in a database for later comparison with future audio inputs. Afterwards, the system had to capture an input from any user and match its spectral coefficients to all previously stored coefficients on the database, in order to identify the unknown speaker. Since the spectral coefficients were acquired adequately the system as a whole did not recognize anything, although we believe that modifying the system's structure to decrease timing dependencies between the subsystems might make the implementation more feasible and less complex.

Table of Contents:

Table of Figures:

1. Design Overview

The overall design methodology for this project was to divide the system into two parts that could be done relatively independently. Unfortunately, the project ended up being more suitable for a three person subdivision and the remaining division had to be distributed as best as we could. Figure 1 shows the whole system's block diagram. As can be seen the architecture used is fairly compact given the nature of the system. As a result of this desired compactness, system complexity was underestimated. In particular, the much desired modularity between our individual parts ended up being completely useless because the distance subsystem and the control unit should have been designed by the same person to facilitate posterior integration.

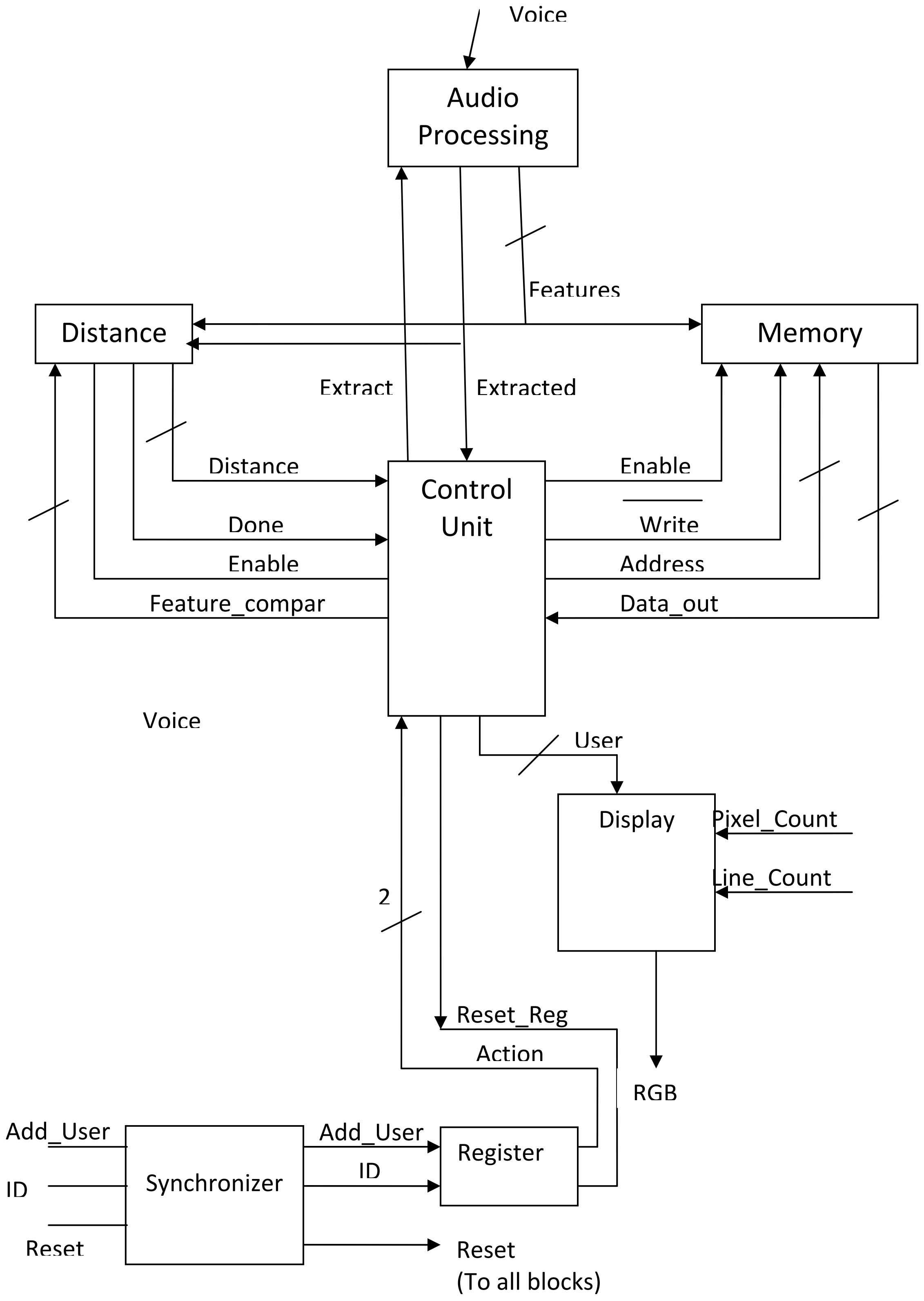

Figure 1. Block Diagram of the Voice Recognition System

2. Module Description and Implementation

2.1 Audio Processing Subsystem (J)

When the Control Unit asserts a signal requesting audio to be processed, this subsystem collects and processes real-time audio signals of approximately 6.8 seconds and determines their spectral coefficients. Figure 2 shows this subsystem's block diagram. There are two main units within this subsystem: the Front End Processing Unit gathers the audio samples that will be processed, and the Back End Processing Unit performs the actual processing.

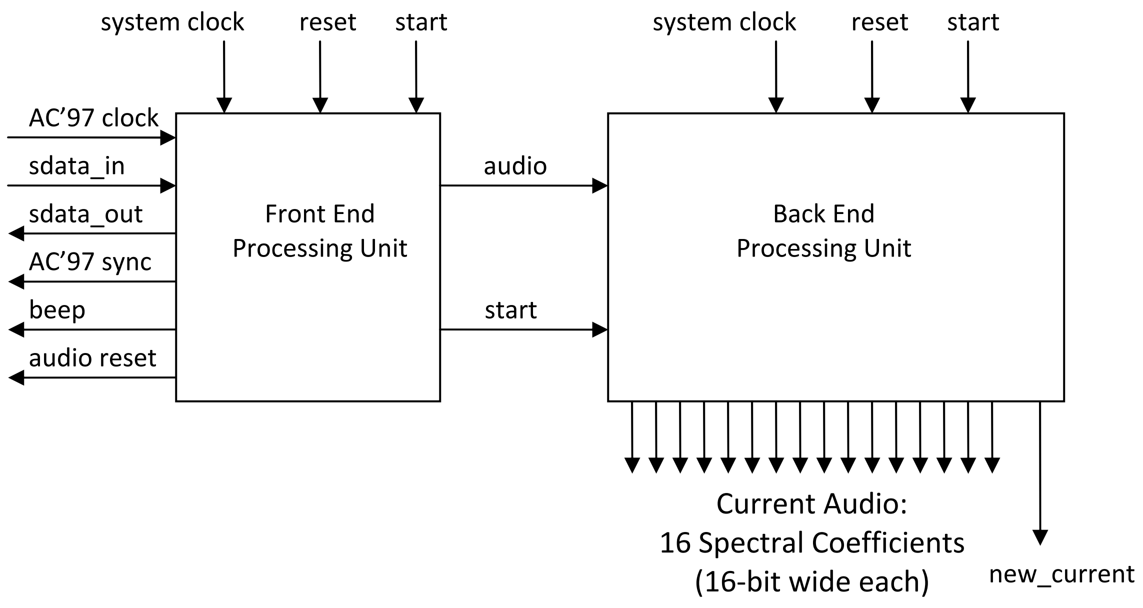

Figure 2. Block Diagram for the Audio Processing Subsystem

2.1.1 Front End Processing Unit

This unit gathers samples of real-time audio from the AC'97 codec at a 48 kHz rate and serially outputs 160 time intervals of 2048 audio samples each whenever the Control Unit requests input audio to be processed. This implies that exactly 6.8267 seconds of audio input is processed each time this subsystem is requested to operate. An important aspect that should not be overlooked is that this unit handles two asynchronous clocks, namely, the AC'97 clock (12.288 MHz) and the system's internal clock (31.5 MHz). The design advantages of this configuration will be explained below. Figure 3 shows the State Transition Diagram for this unit.

2.1.2 AC'97 Controller module

This module takes three inputs: the AC’97 clock, a reset signal, and a one-bit data signal (sdata_in) from the AC’97 containing relevant audio information. Naturally, this module outputs the 18-bit wide audio signal given by the AC’97. However, in order to get such signal the AC’97 synchronization signal must be appropriately outputted to the codec. This would be enough if the AC’97 default were set to output the audio from its microphone ADC. However, the default state of the codec is set to mute all audio, thus an additional output (sdata_in) was properly driven to establish proper volume for our audio input.

In addition to that, the serial data obtained from the AC’97 had to be appropriately read into corresponding 18-bit audio signals. This module implemented a shift register to read the appropriate 18 bits of each frame corresponding to the left audio channel and outputted that value at a 48 kHz frequency along with a 48 kHz 50 % duty cycle clock signal synchronous to the AC’97 clock that is used in the FIFO Buffer module.

2.1.3 Audio Control module

This module’s main purpose is to control the amount of time for which the audio input is processed by signaling the FIFO buffer when to write and read audio signals coming from the AC’97. Upon a start signal assertion from the Control Unit, the module enters a processing state in which initially the write signal is asserted to indicate the FIFO Buffer that it should accept input audio and a counter is initialized to 0. When the FIFO gets full, the counter is increased by 1, the write signal is deasserted and the read signal is asserted. Therefore the FIFO Buffer unloads the 2048 samples that it had previously stored sequentially. When the FIFO asserts the empty signal, the module deasserts the read signal and asserts the write signal again, still remaining in the processing state. This cycle continues until the counter reaches 160. When this happens, the module returns to the IDLE state and is ready for a new processing request from the Control Unit. Figure shows the State Transition Diagram for this module, which closely follows the whole subsystem operation.

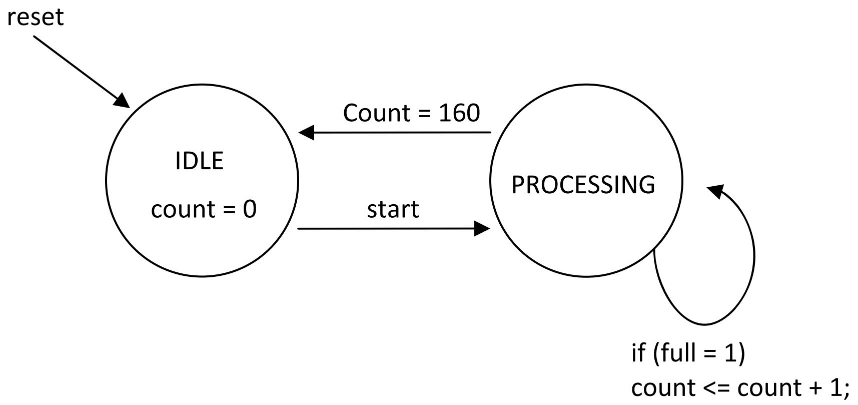

Figure 3. State Transition Diagram for the Audio Control module

2.1.4 FIFO Buffer module

This module is an 18-bit wide by 2048 word long First-In-First-Out (FIFO) buffer with different read and write clocks. Our particular implementation was done using CoreGen’s FIFO buffer. The read clock is the system's 31.5 MHz clock, while the write clock is the 48 kHz signal generated by the AC'97 Controller module. This implies that the FIFO takes a lot longer to fill up than it takes to unload. The data to be written is the 18-bit wide output audio from the AC'97 Controller module, which was designed to be synchronous to the positive edge of the 48 kHz signal. The data that is read is the 18-bit wide audio output of the Front End Processing Unit.

This module takes audio samples from the AC'97 Controller module and stores them temporarily whenever the write signal is asserted by the Audio Controller module and it is not already full. Similarly, it outputs stored audio samples whenever the read signal from the Audio Control module is asserted and the buffer is not empty. Due to the different read and write clocks, data unloading is much faster than data loading. The main advantage of this design is that it guarantees that the output signal will be synchronized to the system's clock, meaning that we solve the asynchronization issue. This in turn enables us to process the data using the system's higher clock frequency so the output coefficients are computed faster and stay stable for a longer period of time, which makes possible our Distance Processor Unit configuration, discussed later.

2.1.5 Back End Processing Unit

After having established correct timing for our input audio, this unit processes the audio data by passing it through the serial network of modules, as shown in Figure 4.

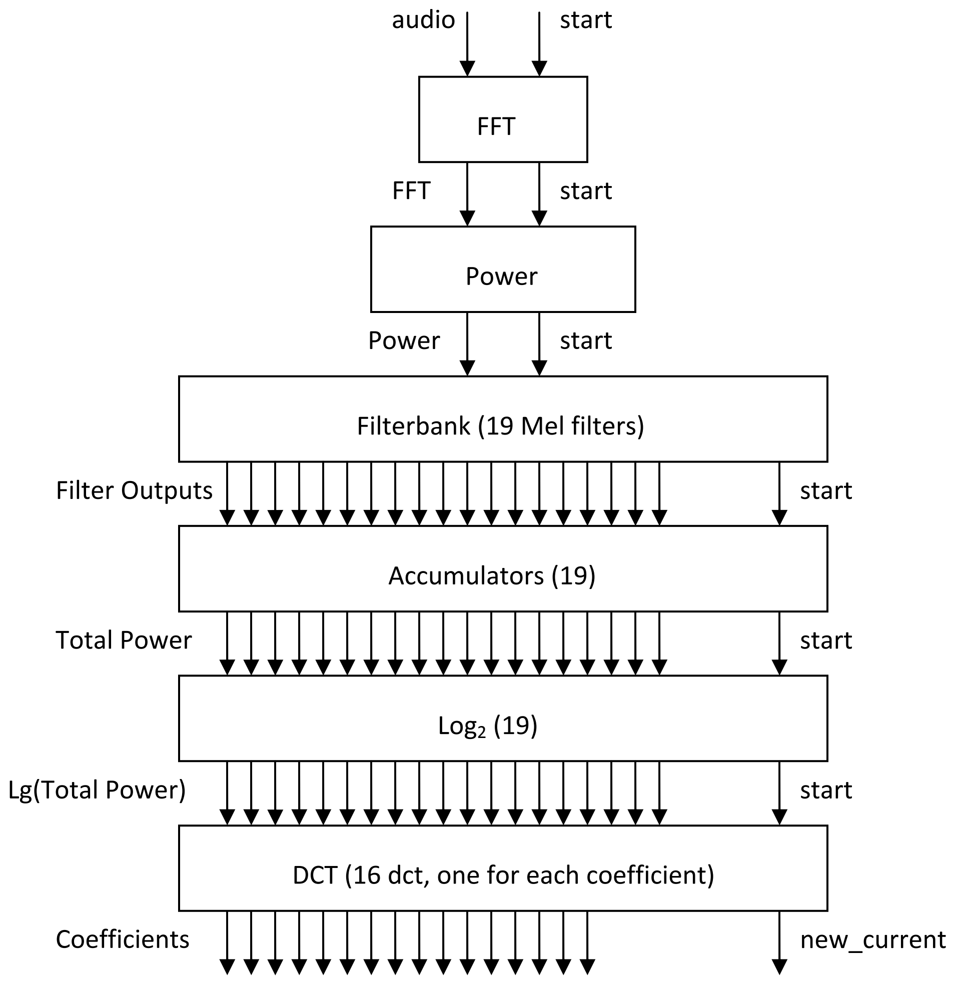

Figure 4. Block Diagram of the Back End Processing Unit

An important design aspect that can be inferred from the architecture is that modules are not arranged in a typical Major-Minor FSM structure. The implemented architecture is based on an all-Minor FSM structure. Such arrangement is not optimal in typical digital systems because it provides less modularity. However, a typical Major-Minor FSM setup for this particular application would not improve modularity either because each data processing sequence is the same. Each block's function in Figure 4 is self-explanatory and detailed explanation of each is beyond the scope of this report, although it must be mentioned that all blocks were implemented manually, with the exception of the FFT which was obtained from a built-in CoreGen.

Overall, this unit had 39 distinct modules (1 FFT, 1 power convertor, 19 filters, an accumulator, a log2 calculator, and 16 DCTs) and 75 total module instances (1 FFT, 1 power convertor, 19 filters, 19 accumulators, 19 log2 calculators, and 16 DCTs). It must be pointed out, however, that the implementation of this unit was not fully functional. While audio signals were correctly modulated up to the filters, the accumulators used were flawed and did not output correct values. Therefore, the spectral coefficients obtained were not correct and this prevented full system functionality.

2.2 Distance Processor Subsystem (J)

A problem that arises when two audio samples divided into equally spaced time intervals are to be matched is that time misalignment of the audio makes it impossible to calculate an efficient Distance Metric using a single comparison between coefficients at the same time intervals. This subsystem uses a simplified adaptation of the Dynamic Time Warping algorithm presented in Kavaler et al. [1] to solve this problem. The main idea behind our implementation is that the current audio's coefficients at any time interval should be at a minimum Euclidean distance from one or more time interval coefficients only for the correct stored user. Similarly, if the spoken audio does not correspond to a specific user, the input audio's coefficients should not be close to any of that user's coefficients at any time interval. This is done for all time intervals of the current input audio, and then the Distance Metric for each user that will be used to distinguish among them is obtained by adding the minimum distances of each user at all 160 time intervals. The only difference between the approach presented in Kavaler et al. [1] and our adaptation is that ours is more flexible and less computationally intensive in the sense that it looks for the coefficients in the time interval that minimizes Euclidean distance without checking that the selected time intervals are in a coherent order.

Our implementation calculates Euclidean distance in a serial manner. That is, it takes as input two sets of 16 spectral coefficients, each set with a corresponding signal that goes high for one clock cycle to indicate when it has changed its value. The Euclidean distance among the two is calculated and additional counters and logic are used to determine and store the minimum distances of each particular user on any current input's time interval and to perform the accumulating function of these minimum distances for all the 160 current input's time intervals. The final outputs of this module are four 48-bit signals, each giving the total distance between the current input's coefficients and the stored user's coefficients, along with a signal indicating that the Distance Metric is valid. Figure 5 shows the complete Block Diagram for this unit.

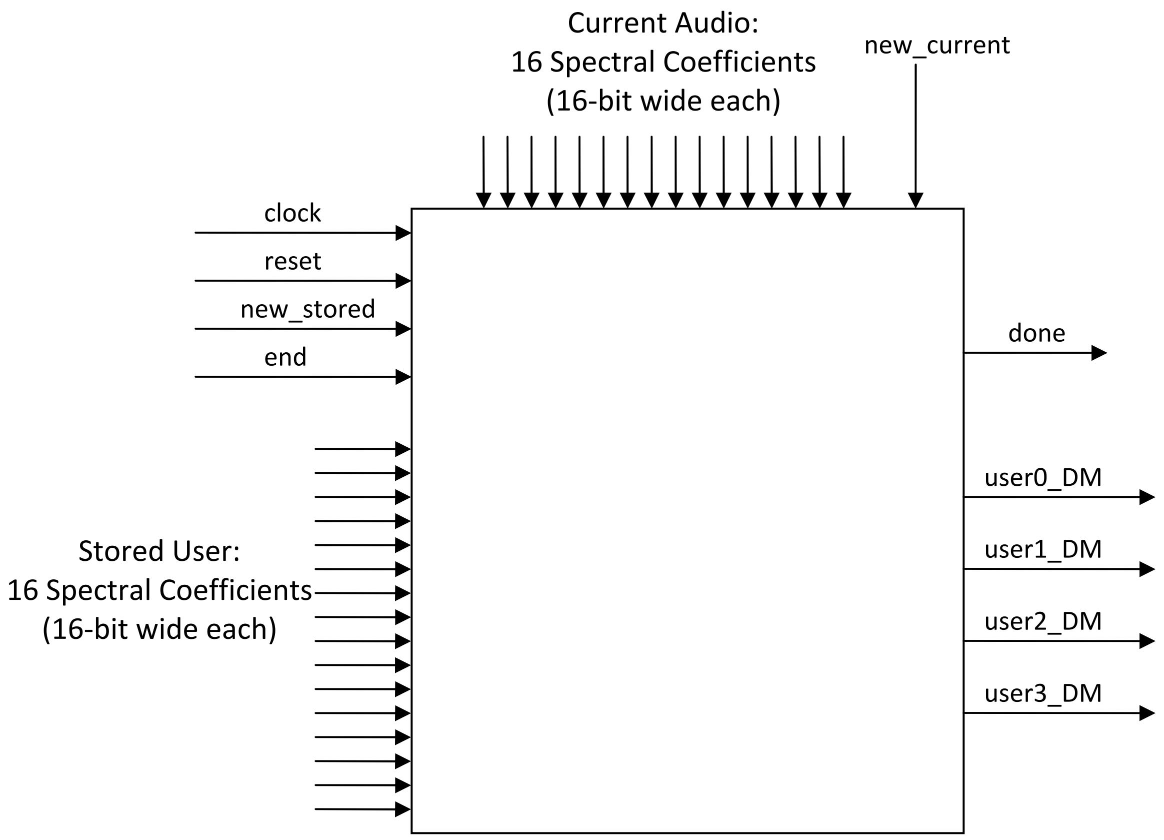

Figure 5. Block Diagram of the Distance Processor Subsystem

Note that 48 bits is way shorter than the width needed for this calculation to be completely accurate, which is (number of additions in the accumulation part, 160) + (width of the Euclidean Distance calculated, 23 bits) = 183 bits. This width reduction worked on the assumption that whenever the current speaker was not speaking, the audio input would completely match all users because all stored users had stayed silent for some period of time during their initial voice input. Thus, all silent intervals would generate very small distances, i.e. for each silent time interval of the current user one MSB of a 208-bit accumulator could be removed.

Expecting this last assumption to be unrealistic, this unit's implementation was based on a partial Major-Minor FSM setup to allow more accurate compression methods to be developed if needed. Indeed this is not a full Major-Minor FSM setup because there is only one Minor FSM, namely, the Vector_Distance module which has own states and computes the Euclidean distance between two 16-dimensinal vectors by using the aid of a customized square root module also implemented. Figure 6 shows the State Transition Diagram for this unit.

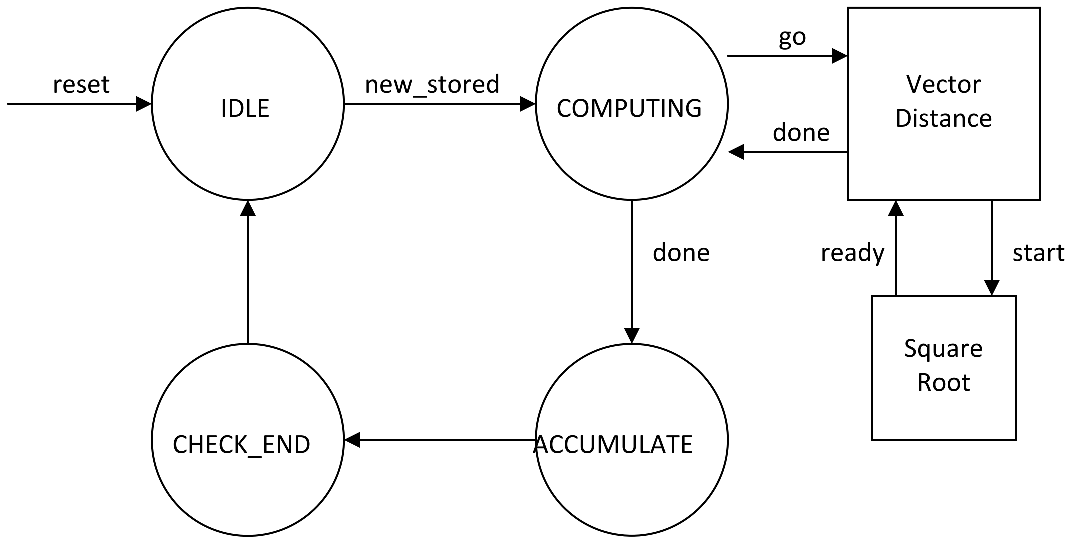

Figure 6. State Transition Diagram for the Distance Processor Unit.

Upon asserting reset, this unit enters an IDLE state in which it does no computations. If the current audio coefficients change, a counter is increased by one to indicate the current audio time interval that is being processed. The Control Unit is expected to provide all the stored user's coefficients after each change in the current audio coefficients. It does so by asserting a signal indicating that a set of 16 stored coefficients is ready and waiting for the processor to go to the IDLE state before asserting the signal again to indicate that a new set of 16 coefficients is ready. This ready signal increases a counter that is used to determine when all the 160 time intervals of a specific user have been loaded, and therefore stores the minimum distance that it has so far for that user at that time interval. It is important to note that inputting stored coefficients as fast as possible into the processor is crucial because when the coefficients of the next current audio time interval come, the processor must have finished all computations for that time interval for this scheme to work. Clearly, this imposes a maximum on the number of users that can be stored, but for our 4-user system this is not a problem because each input audio time interval is generated at under a 24 Hz rate while sets of coefficients can be processed at over 1 MHz (under 30 clocks cycles per set). That is, using our 160 time interval divisions, up to 1 MHz / (24 Hz * 160) = 260 users could be processed without problems if the Control Unit optimizes memory read cycles. This cycle repeats itself for the 160 time intervals of the current audio input and each time the stored minimum distances get added to the previous ones. However, anytime during the computations for last current audio coefficients, the Countrol Unit must assert for one clock cycle an additional input signal to indicate it has no more coefficients to give. By doing this, when the last user's coefficients are finished computing, the overall Distance Metrics for the 4 users are passed to the Control Unit along with an asserted signal indicating that the Metrics are valid.

2.3 Operation (R)

The system has two modes of operation: a user can be added to the database of the system, or an unknown user can be identified based on the entries in the database. The mode is controlled with two buttons. Button 3 [activates] the add mode, and button 4 [activates] the user identification mode.

2.3.1 Adding Users to the Database (R)

To add a user to the database, button 3 must be pressed. If the system is not BUSY performing an action the LED lights show that the system is extracting the voice of the user, and the monitor displays an E demonstrating that the voice extraction is being performed. The user should say a word, which will be his password, into the microphone.

After the signal is extracted, and the spectral coefficients are produced, the Control Unit transitions into the ADD state, in which the coefficients will be stored into the 16 RAMs. While accessing the memory, the LED lights show that the system is adding a new user, and the monitor displays an A, [signaling] that an ADD is been performed.

When the storing action is complete, the LED lights show that the system is done with the ADD, and the monitor displays an AD, stating that an ADD has been performed. Immediately after, the system is ready to perform another action. The Control Unit transitions into the IDLE state, the LED lights show that the system is ready for an action, and the monitor displays READY.

2.3.2 Identifying Users (R)

To identify a user to the database, button 4 must be pressed. Similarly to the ADD mode, if the system is not BUSY the LED lights show that the system is extracting the voice of the user, and the monitor displays and E, signaling the voice extraction. The user should [say] his password into the microphone.

After the signal is extracted, and the spectral coefficients are produced, the Control Unit transitions into the ID state, in which the coefficients for each of the users previously added will be fetched from the database. While fetching the spectral coefficients from the memory, and comparing them with the ones produced with for the user that is [asking] for an identification the LED lights sow that the system id identifying a user, and the monitor displays an I, indicating that an ID is being performed.

When the distance and the validation modules finish comparing the spectral coefficients, and comparing the user distances, respectively, the displays change. The LED lights show that the system is done with the ID, and the monitor displays and ID, confirming that the ID has been performed. Subsequently, the system is ready to perform another action. The Control Unit transitions into the IDLE state, the LED lights show that the system is ready for an action, and the monitor displays READY.

2. 4 Control Unit (R)

The Control Unit module is the [big picture] controller for the system. This module has five states: IDLE, RESET, BUSY0, BUSY1, and BUSY2. Figure7 provides a representation of these states. It works as a major FSM and controls the LED lights, provides the input for the display module, instantiates the minor FSMs: MEM_ID, MEM_ADD, and VALIDATION.

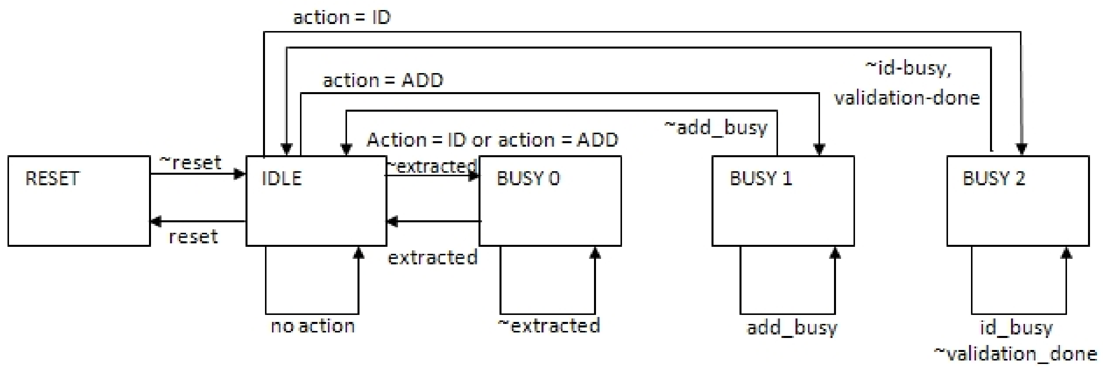

Figure 7. Control Unit Major FSM State Diagram.

The IDLE state is the ready mode, in which the system is waiting for an action. IN this mode the user can select to either add or identify a user, utilizing button3 and button4, respectively. When the user presses any of these two buttons, the controller will transition to the BUSY0 state. At this state a pulse is sent to the voice module via the extract signal. The module remains in this state until the voice module sends a pulse via the extracted signal, indicating that the voice extraction and analysis has been performed. When the extracted pulse is received, the controller will transition to the BUSY1 if the user selected the ADD mode, and into BUSY2 if the user selected the ID mode.

For adding a user to the database, the controller remains at the BUSY1 state until the MEM_ADD minor busy signal goes low. For identifying a user, the controller will remains at the BUSY2 state until the MEM_ID minor busy signal goes low, and the VALIDATION minor send a pulse via the validation_done signal. After completing any of the two actions the system will return to the IDLE state.

If the user desires to reset the system, button0 should be pressed. The Control Unit module will transition into the RESET state, reset its outputs, and instruct the rest the rest minor FSMs to reset their outputs.

2.5 ADD: (R)

The MEM_ADD module is one of the minor FSMs instantiated by the Control Unit. This module has three states: IDLE, PROCESSING1, and PROCESSING2. Figure8 provides a representation of these states. When the Control Unit sends a pulse via the add_start signal, the MEM_ADD module transitions from the IDLE state to the PROCESSING1 state. While transitioning, the module will activate the write-enable, produce the memory address signal, and start sending the features, produced by the voice module, to the memory.

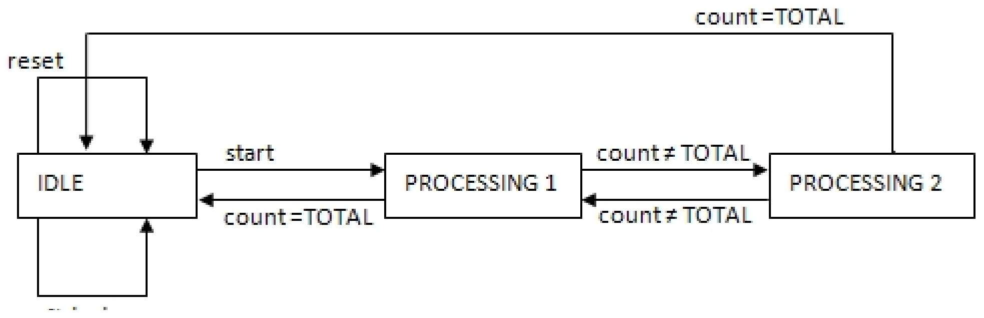

Figure 8.MEM_ADD Minor FSM State Diagram.

The module will transition repeatedly from PROCESSING1 to PROCESSING2, which is identical to PROCESSING1, and vice versa, until the internal count reaches 159. Because 160 is the amount of features received for each user. At that moment sends the last features to the memory and transition into the IDLE state.

2.6 IDENTIFY: (R)

The MEM_ID module is another minor FSM instantiated by the Control Unit. This module has three staes: IDLE, PROCESSING1, and PROCESSING2. Figure 9 provides a representation of these states. When the Control Unit send a pulse via the id_start signal, the MEM_ID module transitions from the IDLE state to the PROCESSING1 state. While transitioning, the module will produce the memory address signal, and will start [fetching] the stored features from the memory. When the module receives the features from the memory, it sends the data to the Distance module.

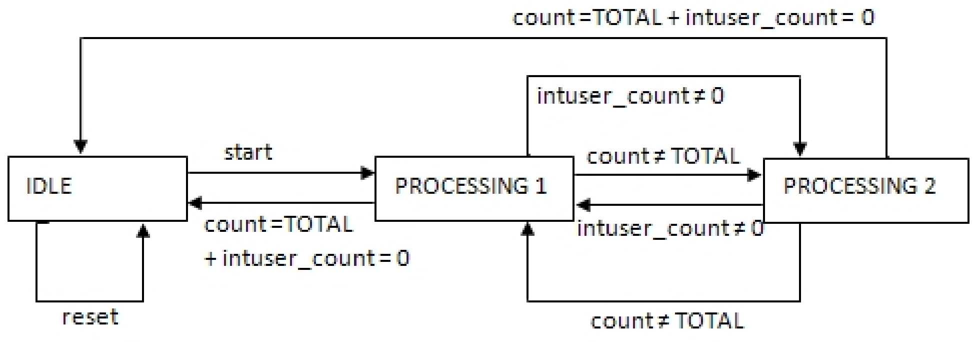

Figure 9. MEM_ID Minor FSM State Diagram.

The module will transition repeatedly from PROCESSING1 to PROCESSING2, which is identical to PROCESSING1, and vice versa, until the 159 features for each of the users have been [fetched]. At that moment the module sends the last features to the memory and transition into the IDLE state.

2.7 VALIDATION: (R)

The VALIDATION module is another minor FSM instantiated by the Control Unit. This module has two states: IDLE, and PROCESSING. Figure 10 provides a representation of these states. When the Control Unit sends a pulse via the validation_start signal, the VALIDATION module transitions from the IDLE state to the PROCESSING state. At the PROCESSING state, the module compares the user's distances provided by the distance module and selects the user with the smallest distance.

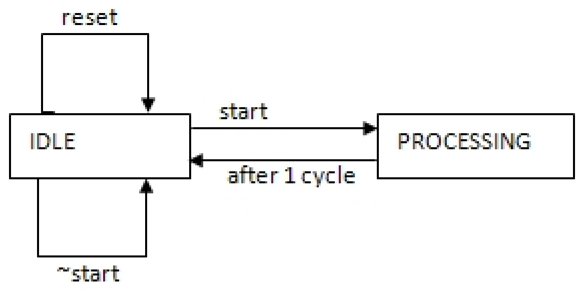

Figure 10. VALIDATION Minor FSM State Diagram.

2.8 Memory: (R)

In order to store and read the features produced by each voice input in an efficient manner, sixteen 16x640 RAMs were produced, one RAM for each of the spectral coefficients. The RAMs were [produced] utilizing Coregen, and instantiated in the labkit.

2.9 Action Register: (R)

The ACTION_REG is a module responsible of receiving the synchronized user inputs, ADD or ID. If the current system is not performing an action, the register will send the selected action to the Control Unit via the action signal.

2.10 VGA: (R)

The VGA module is responsible for producing the vertical and horizontal sync and blanking signals and for producing a pixel and line count.

2.10.1 Display: (R)

The Display module is responsible for producing the red, green, and blue signals for the video. It receives the pixel and line counts from the VGA, and a vgaouput signal from the Control Unit that encodes the text that it must produce. To produce the text the module instantiates another one called rectangle.

2.10.2 Rectangle: (R)

The Rectangle module is responsible for producing rectangles that are used by the Display Field module to display the text. It receives the pixel and line count produced by the VGA, and produces the rectangles that create the text.

3. Testing and Debugging:

Overall, ModelSim simulations were the preferred method of testing individual modules because it is much quicker. However, since total system integration was not possible, to actually prove that the implementations were correct, the subsystems were also tested with FPGA implementations using additional modules that simulated input behavior.

3.1 Simulated Waveforms: (R)

The Control Unit and the minor FSMs were tested by creating Verilog test benches within the project itself. ModelSim generated waveforms that were utilized by the programmer to study the behavior and functionality of the modules. Test benches were created for testing the Control Unit, the Add FSM, the Identification FSM, the Validation FSM, the Memory, and the Action Register.

3.2 Audio Testing: (J)

In the case of the Audio Processing subsystem, this was not necessary because the input audio already came from the FPGA, although the start signal was simulated as a standard input button that we could press. The proper output was then analyzed using the digital oscilloscope and appropriate triggering methods. From these tests it was discovered that the actual outputs on the accumulators were not what they should be although the path up to the filter outputs seemed fine. The problem could not be solved.

3.3 Distance Testing: (J)

Meanwhile, to test the Distance Processor subsystem, a similar approach was taken, a testbench that simulated constant input audio spectral coefficients and two different sets of constants stored coefficients was generated. The module that generated this is included in the Appendix. Upon pressing the start input button, a state toggled and one of two stored sets of spectral coefficients was chosen. This remained constant throughout the whole sequence. The test bench then cycled through all the necessary start/ready loops and finally signaled the processor that no more data was stored. The observed output showed correct values for each of the two sets of stored coefficients.

4. Conclusion:

The objective of this final project was to design and implement a complex digital system combining voice, video and user interfaces. The analysis presented in the previous sections shows a fully functional control units, video displays, user interfaces, and voice extraction system. In addition, it includes a fully functional voice modulation algorithm, and a partially functional voice modulation system. A comprehensive testing and debugging methodology was utilized, which validated the functionality of the different modules.

Overall this project could have been a great success. However, due to time constraints and the complexity of what the team wanted to do impeded the integration of all modules and the completion of our project.

Some of the lessons learned from this project are the importance of a good design planning and long hours of testing and debugging. As a whole, this project gave insight into designing a complex system which could be divided and tested separately. If we would have been able to integrate the whole project, this would have made the integration portion easier.

5. References

[1] Kavaler et al. A Dynamic-Time-Warp Integrated Circuit for a 1000-Word Speech Recognition System IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. SC-22, NCI. 1, FEBRUARY 1987

6. Appedinx

6.1 Simulations

6.2 Code