

Speech Recognition Using FPGA Technology

	Authors:	
Carlos Asmat	260148251	_____
David López Sanzo	260146414	_____
Kanwen Wu	260045745	_____

Design Project Laboratory
ECSE 494

Submitted on June 18, 2007



McGill

Department of Electrical Computer and Software Engineering

Abstract

An increasingly popular way to interact with machines is to simply talk to them. However, there is often a trade-off between ease of use and system complexity. Thus, the main objective of this project is to design and implement a speech recognition system using a Field Programmable Gate Array (FPGA). It must be capable of accurately identifying a single sound while remaining simple and fast. For this purpose, an algorithm is prototyped and tested using MATLAB™, before being implemented on an Altera DE2 board. Complications in acquiring a Fast Fourier Transform (FFT) module lead to a partial, yet functional hardware implementation.

Acknowledgments

The authors would like to express their gratitude towards Professor Miguel Marin, who provided guidance throughout the realization of this project. They would also like to acknowledge the great interest Professor Richard Rose showed by answering questions, especially regarding the FFT. Finally, they thank Professor James J. Clark for his guidance in the choice of a development board.

Table of Contents

Introduction.....	1
Chapter 1:Background Theory.....	2
1.1 Speech Recognition Principle.....	2
1.2 Data Acquisition.....	2
1.3 Detection.....	2
1.4 Frequency Content.....	3
1.5 Distance.....	4
Chapter 2:Hardware Implementation.....	5
2.1 Wolfson Interface.....	5
2.1.1 I2C Bus Controller.....	5
2.1.2 Sound fetcher.....	6
2.1.3 Clock Module.....	8
2.2 Detector.....	9
2.3 FFT.....	10
2.4 Memory Management.....	10
2.4.1 Memory Controller.....	11
2.4.2 Memory Batch Operator.....	12
2.5 Distance.....	13
2.6 System Controller.....	14
Chapter 3:Results.....	15
3.1 FFT Issues.....	15
3.2 MATLAB Results.....	15
3.3 Experimental results.....	17
Conclusion.....	19
Bibliography.....	20
References.....	20
Appendix.....	21
Appendix B.....	24
Appendix C.....	27
Appendix D.....	28
Appendix E.....	29

Illustration Index

Figure 1: Two-line I2C bus protocol for the Wolfson WM8731.....	5
Figure 2: ADCDAT output convention used by the Wolfson WM8731 (I2S).....	6
Figure 3: Circuit schematic of the overall ADCDAT fetcher.....	7
Figure 4: Block diagram of the clock module.....	8
Figure 5: Block diagram of the word detector datapath.....	9
Figure 6: 512 kB SRAM chip block diagram.....	11
Figure 7: Memory Controller block diagram.....	11
Figure 8: Memory Batch Operator block diagram.....	12
Figure 9: Distance module block diagram.....	13
Figure 10: Overall System Diagram.....	14

Introduction

Speech recognition is becoming increasingly popular and can be found in luxury cars, mobile phones, dictation office software, toys, and appliances. It is defined as the ability to identify a spoken word or a sequence of words. This project is focused on a single-word speech recognition system implemented on an FPGA. It needs to be accurate, reliable, and simple in order to lay a basis for future consumer products. The main idea behind the system, is to first train it with several versions of the same word, thus yielding a “reference fingerprint”. Then, subsequent words can be recognized based on how close they are to this fingerprint. This technique requires some background theory, as depicted in Chapter 1. This research, followed by a MATLAB algorithm prototype leads to the hardware implementation of the system on an Altera DE2 board. The latter is shown in Chapter 2. Finally, the results are discussed in Chapter 3.

Chapter 1: Background Theory

1.1 Speech Recognition Principle

Speech recognition is performed by identifying a sound based on its frequency content. In order to achieve this, the frequency content of several samples of the same sound must be averaged in a training phase (i.e. the sound's “reference fingerprint” must be generated). Then, the frequency content of a sound input can be compared to the aforementioned fingerprint by treating them as vectors and computing the distance between them. If a sound is close enough to the reference, then it is considered to be a match. A MATLAB implementation of this process was created in order to better illustrate it, and experiment with the settings.

1.2 Data Acquisition

Sound waves are analog in nature. Thus, they must be converted to binary data in order to be read and processed by a digital system. This task is performed by an analog-to-digital converter (ADC). It maps the waves' amplitude to a binary number at a given frequency (the sampling frequency). The precision and the sampling frequency required for the scope of this project are respectively 8 bits and 5 kHz. The former has been chosen for convenience (after making sure it was sufficient), while the latter is the standard used by telephone lines.

Note that, for the MATLAB script, the data is input using WAV files containing recorded words.

1.3 Detection

The system must know when a spoken word is input. Thus, a detection algorithm has been devised. This is done by continually computing the difference of the absolute average of two adjacent sound windows (sets of consecutive sound data), and comparing it to a predefined threshold.

The detector algorithm can be broken down as follows:

1. The absolute average w_1 of a sound window of length W is computed from the sound samples s_i starting at s_a and ending at s_b as shown in Eq. 1.

$$w_1 = \frac{1}{W} \sum_{i=a}^b |s_i| \quad (1)$$

2. The average of the second window w_2 is computed from the sound samples s_i starting at s_b and ending at s_c as shown in Eq. 2.

$$w_2 = \frac{1}{W} \sum_{i=b}^c |s_i| \quad (2)$$

3. The difference between w_2 and w_1 is compared to the threshold value Th . If it is larger, the spoken word is considered to start at s_c . Else, the algorithm goes on to step 4.
4. The average of the oldest window (w_1) is discarded, and replaced by w_2 . Then, the algorithm goes back to step 2.

Note that the Th value has been experimentally determined in the MATLAB implementation (see appendix A). Nevertheless, it may vary depending on the sound acquisition setup (i.e. position of the microphone, noise level, etc.). Finally, the length of the word is fixed to 1.024s for convenience.

1.4 Frequency Content

Once the word is detected, it is mapped to the frequency domain by computing its Discrete Fourier Transform (DFT) using the Fast Fourier Transform (FFT) algorithm. Since the length of a word is 1.024 s and the sound is sampled at 5 kHz, five 1024-points FFTs are required to fully characterize a single word. In the MATLAB implementation, these are stored in each row of a 1024 x 5 matrix. This matrix constitutes the “fingerprint”. Note that, for the sake of simplicity, only the real part of the DFT is kept.

In the training mode, the user defines how many times a word is trained. The frequency content of each is averaged by adding their fingerprints together and dividing the final sum by the number of

times the word has been trained. This generates the “reference fingerprint”.

1.5 Distance

The comparison between a word's fingerprint and the reference fingerprint is done by taking the euclidean distance between them. To do this, they are considered as five 1024-dimensional vectors (one for each matrix row), and the average of their respective euclidean distance is computed. This is shown in Eq. 3, where D is the distance, and a_{ni} and b_{ni} are the i^{th} components of the fingerprints. The n index points to each of the five vector pairs.

$$D = \frac{1}{5} \sum_{n=1}^5 \sqrt{\sum_{i=1}^{1024} (a_{ni} - b_{ni})^2} \quad (3)$$

If the distance is less than a preset maximum (maxDis), then the analyzed word is considered to match the reference word. Note that maxDis is experimentally set to 140 in the MATLAB implementation (see appendix B). Similarly to the T_h parameter, this value depends on the sound acquisition setup and may need to be varied in order to achieve accurate speech recognition.

Chapter 2: Hardware Implementation

In order to implement the speech recognition algorithm in the Altera DE2 board, it is broken down into modules. These are then mapped to combinational logic and finite-state machines (FSM), using the Quartus II software package.

2.1 Wolfson Interface

The board has a Wolfson WM8731 Coder-Decoder (CODEC), which acts as the ADC. This audio chip has a microphone jack, and is connected in a master-slave configuration with the FPGA (the latter being the master). In order for the master to control the CODEC and acquire the digital data, three modules have been created: the I²C bus controller, the clock module, and a sound fetcher.

2.1.1 I²C Bus Controller

Three tasks need to be performed on the CODEC to modify its internal settings: “de-mute” the microphone input, boost the microphone volume, and change the default sound path (so that the microphone is given priority over other inputs). To do this, the FPGA communicates with the Wolfson via the I²C (Inter-Integrated Circuit) protocol using two pins: 'SDIN' (the data line), and 'SCLK' (the bus clock), as seen in Fig. 1.

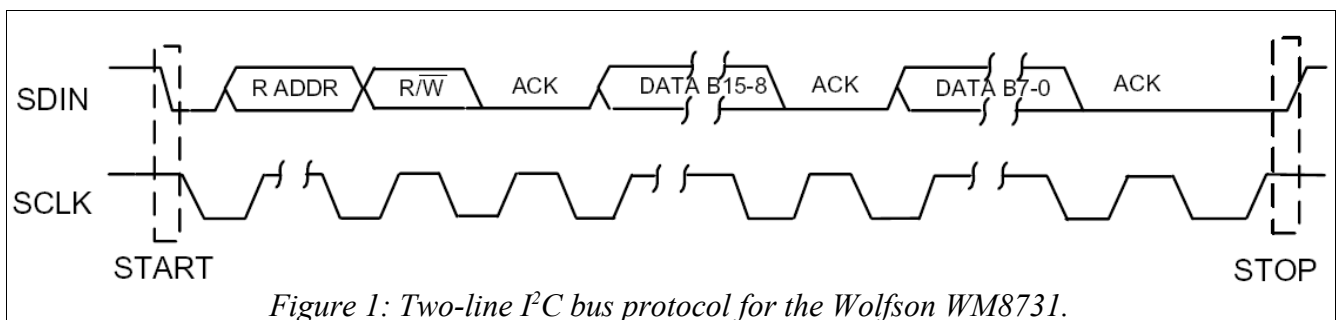


Figure 1: Two-line I²C bus protocol for the Wolfson WM8731.

The contents of the data line are sent in the same order as seen above (after a start condition):

'RADDR', 'R/W', 'ACK', 'DATAB[15-9]', and 'DATAB[8-0]', which stand respectively for “base address”, “Read/Write”, “acknowledge”, “control address”, and “control data”. The last block modifies the settings. For instance, if 'DATAB[0]' is '1', the volume is boosted. The base and control

addresses are used to specify which internal CODEC registers need to be accessed. “Read/Write” will always be set to zero (i.e. write), since the Wolfson is write-only.

To signify a start condition, 'SDIN' goes from high to low while the clock is maintained high. The same applies for a stop condition, except the transition is low-to-high. Finally, the 'ACK' signal is sent from the CODEC to the FPGA, as opposed to all the other data line contents. This introduces the need for 'SDIN' to be implemented as a bi-directional pin, which requires the use of a tri-state buffer. An FSM is created to implement the bus interface between the FPGA and the Wolfson. Note that, because 'SCLK' must be between 0 Hz and 400 kHz, 'ADCLRC' (48.83 kHz) is used (see section 2.1.3). For start and stop conditions, 'ADCLRC' is overridden by the FSM, so that 'SCLK' remains stuck at '1'.

2.1.2 Sound fetcher

After the Wolfson digitalizes the input, it presents the data ('ADCDAT') serially as seen in Fig. 2.

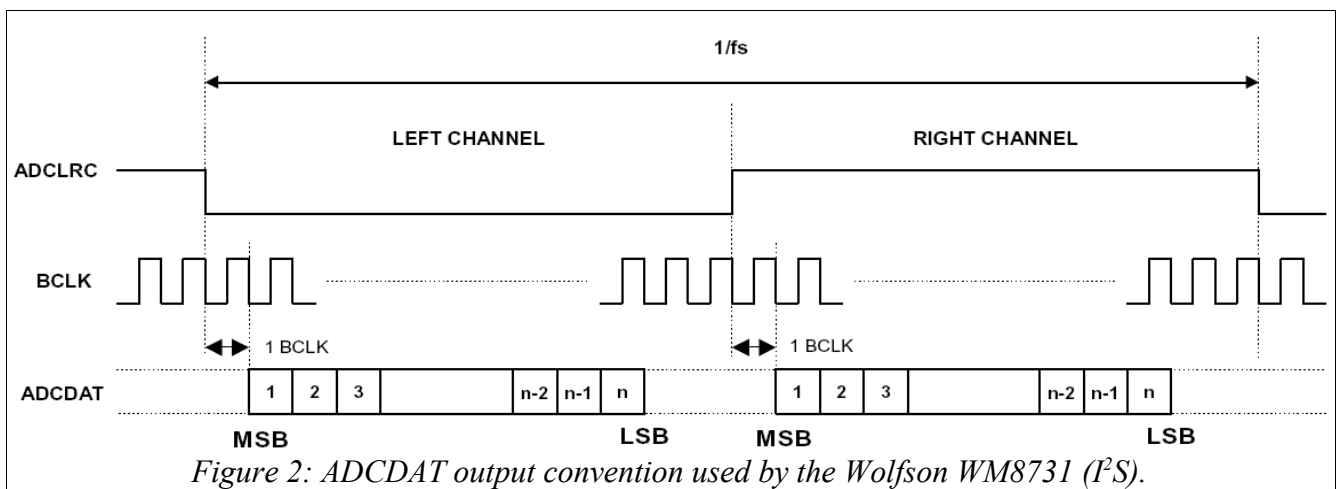
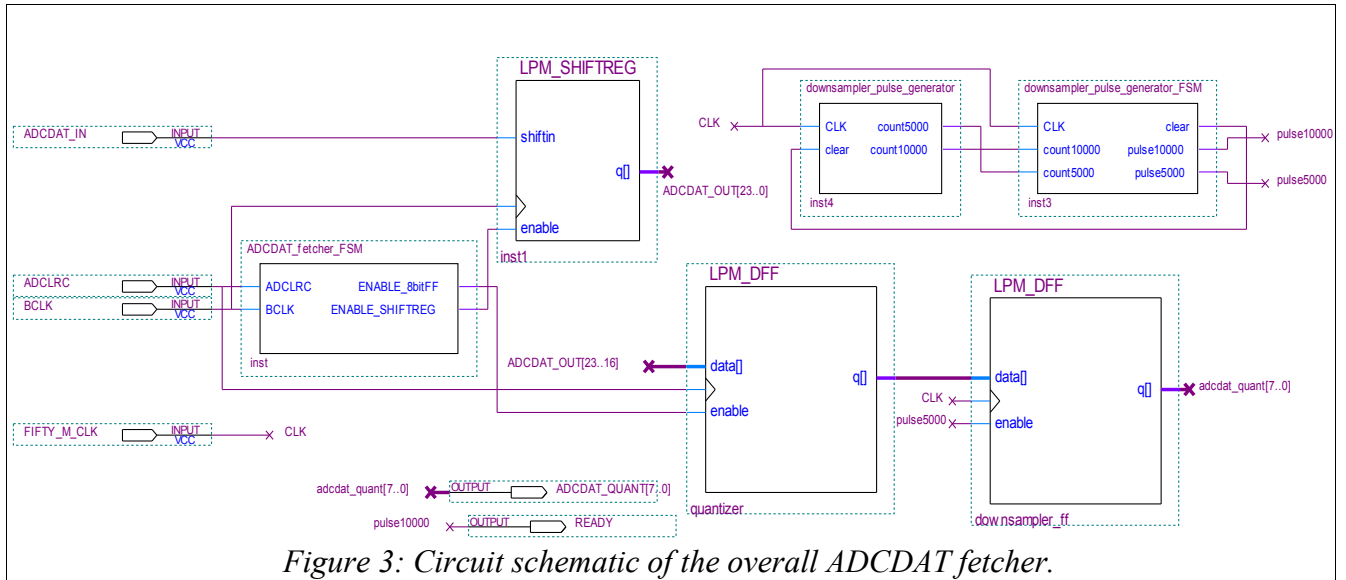


Figure 2: ADCDAT output convention used by the Wolfson WM8731 (I²S).

This is the Integrated Interchip Sound (I²S) standard. Two clocks are needed: 'ADCLRC' (the left-right clock for ADC data), and 'BCLK' (the bit-stream clock). The CODEC will place the most significant bit (MSB) on the 'ADCDAT' line so that it can be fetched on the second rising 'BCLK' edge following a high-to-low transition of 'ADCLRC'. The left and right channel distinction is used for stereo sound. Since this project deals with mono sound, the data is fetched when 'ADCLRC' is low (left channel).

The FSM in Fig. 3 ('ADCDAT_fetcher_FSM') is used to keep track of the events on the clocks (e.g. rising edges) in order to know the exact moment one can start and stop to fetch. Because the data is presented serially, the FSM communicates with a serial-to-parallel register ('LPM_SHIFTREG'), which outputs this data in parallel form.



The next step is to quantize. The 'ADCDAT' word length is 24 bits in two's complement form. As said in section 1.2, the objective is to reduce the length to 8 bits. In order to see how signed binary numbers can be quantized, Table 1 illustrates a quantization from 3 bits to 2 bits.

Decimal number	Binary (2's comp.)	Quantized decimal	Quantized binary (2's comp.)
3	011	1	01
2	010		
1	001	0	00
0	000		
-1	111	-1	11
-2	110		
-3	101	-2	10
-4	100		

Table 1: Two's complement quantization from 3 bits to 2 bits.

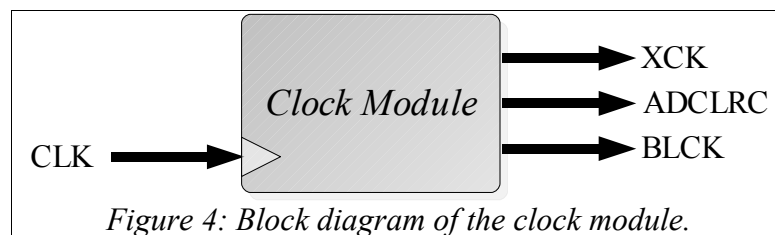
A closer look at the second and fourth columns reveals that, in order to quantize, it is only necessary to keep the two MSBs. Note that this is possible because the two's complement scheme is used.

Consequently, when going from 24 bits to 8 bits, only the first eight most significant bits need to be kept. This is exactly what the D-type flip-flop 'LPM_DFF/quantizer' does in Fig. 3.

The last D-type flip-flop ('LPM_DFF/downsampler_ff') reduces the output data rate from 48 kHz to 5 kHz. In order to do that, it is controlled by the two modules (a counter and an FSM) in the top right corner of Fig. 3, which generate two pulses. Both pulses occur at a 5 kHz frequency. The first instructs the flip-flop to fetch the data. The second pulse is an output 'READY' signal that happens half-a-period after the first. Its purpose is to make sure that the rest of the circuit will fetch the data after it has been properly latched.

2.1.3 Clock Module

The FPGA is clocked at 50 MHz [1]. Because it acts as the Wolfson's master, it must feed the latter with various clocks: the main audio chip clock ('XCK'), 'ADCLRC', and 'BCLK'. According to the Wolfson data sheets, both 'ADCLRC' and 'XCK' are dependent on the sampling frequency. Since the latter is 48 kHz, 'ADCLRC' must also be 48 kHz (as seen in Fig. 2). 'XCK' is 12.288 MHz [4]. 'BCLK' must be at least 2.4 MHz, because it needs to yield 25 rising clock edges (1 to wait for the MSB and 24 to fetch each 'ADCDAT' bit) within half the period of 'ADCLRC' (i.e. within 10.42 μ s).

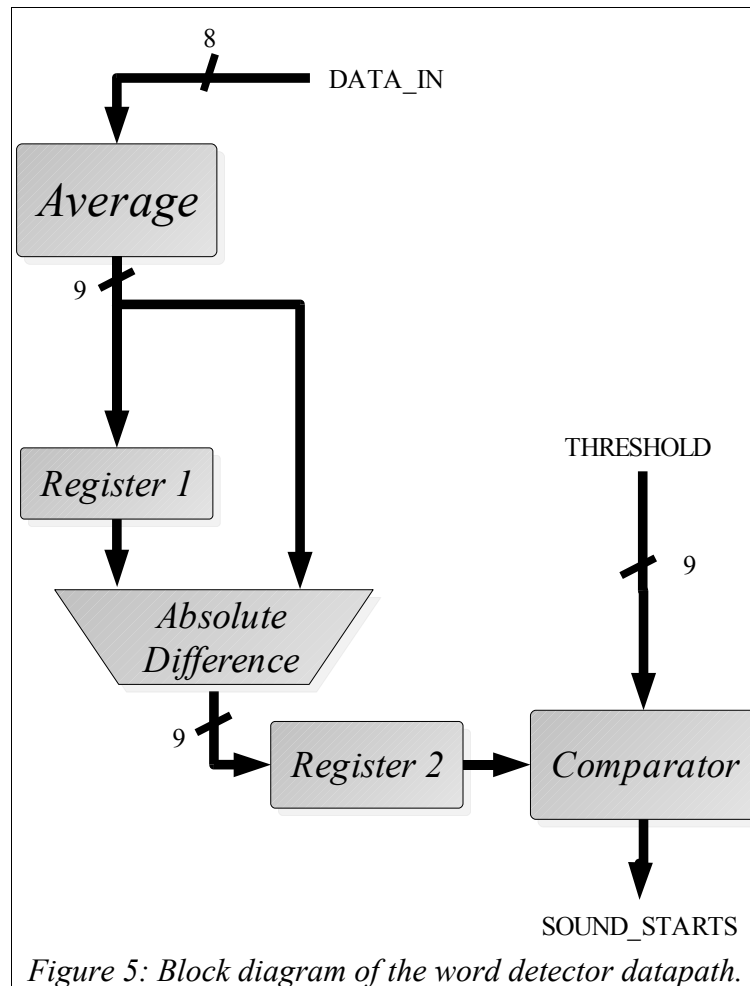


To implement all three clocks, a single clock module was devised. As seen in Fig. 4, it takes the 50 MHz clock as an input. Using a 2-bit counter, it then proceeds to divide it by 2^2 yielding a 12.5 MHz 'XCK' signal. Similarly, 'ADCLRC' and 'BCLK' are output using respectively 10-bit and

3-bit counters (to divide by 2^{10} and 2^3). This produces 48.83 kHz, and 6.25 MHz signals (the latter being greater than 2.4 MHz). Even though those values are approximations of the ideal ones specified in the data sheets, they are close enough for practical purposes [3].

2.2 Detector

The word detector is implemented with the datapath shown in Fig. 5.



The absolute values of the first 1024 samples that constitute a window are accumulated (summed together). Then, the sum is shifted right by 10 in order to divide by 1024 (since $2^{10} = 1024$), thus producing the average value of the window. The absolute value of the difference between that average and the one from the previous window (stored in 'Register 1') is then computed. 'Register 2' is used to control the comparator's input in order to ensure the comparison with a user-defined 9-bit threshold takes place when all the samples of the window have been processed. Once done, the contents of

'Register 1' are replaced by the newer window average.

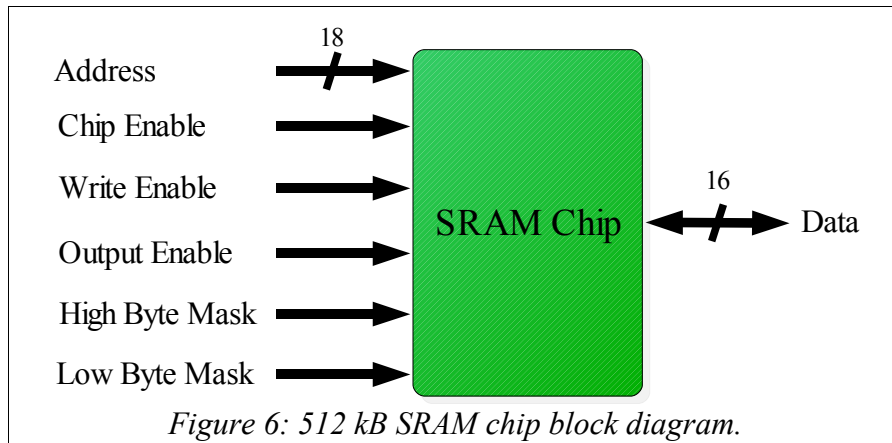
An FSM is needed in order to control when to do this average swapping, when to enable 'Register 2', when to determine if a count of 1024 samples has been reached, and when to clear the accumulator to restart the summation. It also accepts a 'RESET' signal that asynchronously clears the accumulator.

2.3 FFT

In order to map the sound data from the time domain to the frequency domain, the Altera IP Megacore FFT module is used. The module is configured so as to produce a 1024-point FFT. It is not only capable of taking a streaming data input in natural order, but it can also output the transformed data in natural order, with a maximum latency of 1024 clock cycles once all the data (1024 data samples) has been received.

2.4 Memory Management

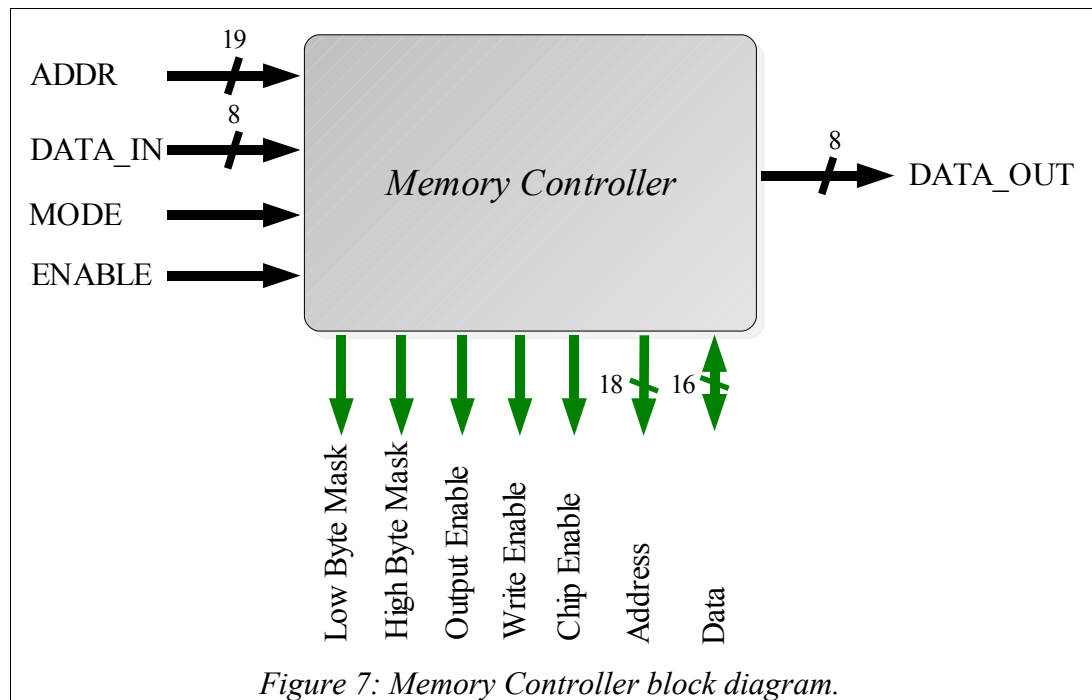
In order to store the reference fingerprint, the 512 kB SRAM module built in the board is used. There are three memory modules on the Altera DE2: a 4 MB Flash memory chip, an 8 MB SDRAM chip and a 512 kB SRAM chip. While the Flash module provides a vast amount of non volatile storage, it is very slow with respect to the main system clock. It also requires a controller capable of dealing with its timing constraints. The SDRAM chip is very fast and has a very large storage capacity, but it requires a very sophisticated controller to be operated. This makes the SRAM chip an obvious choice. Even though it is not the fastest nor the largest, it has ten times the required storage capacity needed for this project, and it is fast enough (since it can perform a read or write operation in less than 20 ns, i.e. a system clock period) so as to avoid any timing issues. Moreover, it is a fairly simple device and can be easily controlled.



The SRAM memory module is depicted in Fig. 6 with its inputs and outputs. Note that the 'Data' pins are bidirectional and require a tristate buffer to be properly driven.

The chip storage is divided into 2^{18} 16-bit blocks which can be directly addressed through the 18 'Address' lines. This is not convenient for the implementation since the data stored is 8-bit wide.

2.4.1 Memory Controller



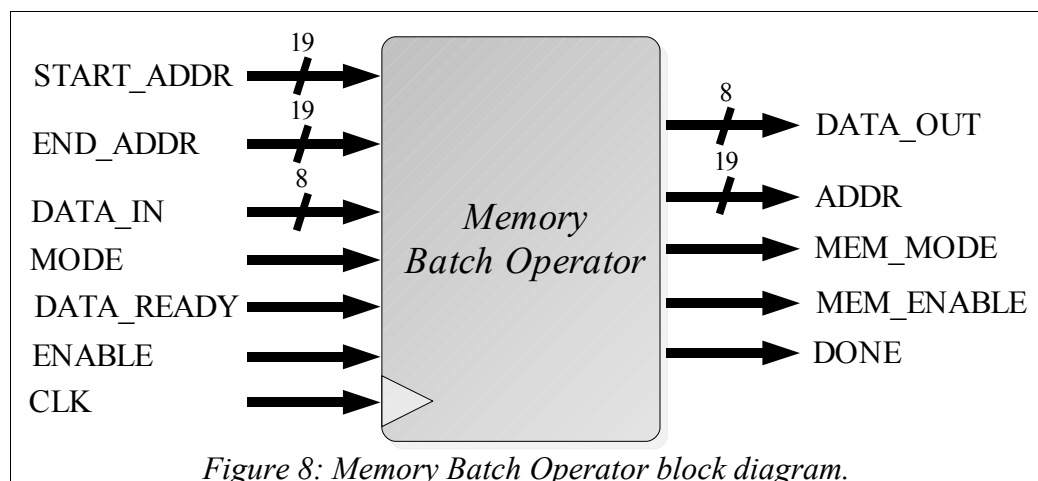
The “Memory Controller” shown in Fig. 7 has four user inputs ('ADDR', 'DATA_IN', 'MODE', and 'ENABLE'), one user output ('DATA_OUT') and seven inputs/outputs (depicted in green) that connect

directly to the SRAM chip ('Low Byte Mask', 'High Byte Mask', 'Output Enable', 'Write Enable', 'Chip Enable', 'Address', and 'Data'). The controller simplifies the communication to the SRAM chip by splitting the bidirectional pins and allowing each 8-bit memory block to be directly accessed (see its detailed schematics in Appendix C). The pins are split by using Altera's "bustri" (tri-state buffer) and each 8-bit block can be accessed using the 'High Byte Mask' and the 'Low Byte Mask' according to the least significant bit of 'ADDR'. As a result, the user sees an 8-bit data input ('DATA_IN'), a separate 8-bit data output ('DATA_OUT') and 19 address lines ('ADDR') which double the original address space.

The memory controller works asynchronously (just as the SRAM chip does) and assumes the SRAM timing constraints are respected. This assumption can be safely made for two reasons: the "read" and "write" operations are completed in 10 ns to 17 ns [2], and the maximum speed at which the chip is accessed is dictated by the main clock (since the entire system works synchronously). This means the memory could never be accessed more than once in 20 ns.

2.4.2 Memory Batch Operator

In order to sequentially access the memory, a 'Memory Batch Operator' module was devised. As shown in Fig. 8, it takes 6 inputs ('START_ADDR', 'END_ADDR', 'DATA_IN', 'MODE', 'DATA_READY', and 'ENABLE') and has 5 outputs ('DATA_OUT', 'ADDR', 'MEM_MODE', 'MEM_ENABLE', and 'DONE'). It operates on the rising edge of a clock signal ('CLK').

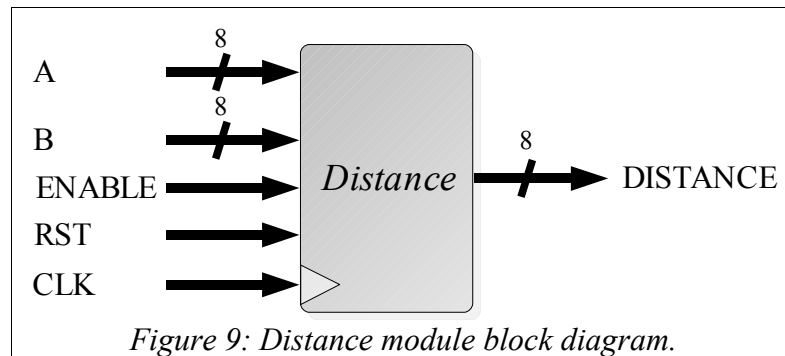


The module works as follows:

1. Whenever the 'ENABLE' input goes high, it fetches the starting and ending addresses as specified in the 'START_ADDR' and 'END_ADDR' inputs, and readies to start writing or reading (according to the 'MODE' input) at the starting address. This takes two clock cycles.
2. Whenever the 'DATA_READY' signal is asserted, the module goes to the next address and reads (the data can be read from the 'DATA_OUT' lines of the memory controller) or writes (the data from the 'DATA_IN' input lines).
3. If the module reaches the ending address, then it signals 'DONE' until the 'ENABLE' input is low and goes back to step 1. Else, it goes back to step 2.

Note that on each step, the module takes care of sending the appropriate signals to the memory controller in order to perform the desired action.

2.5 Distance



The distance module illustrated in Fig. 5 has four inputs ('A', 'B', 'ENABLE', and 'RST') and one output 'Distance'. It computes the distance between two arbitrarily sized vectors by adding and accumulating the squared difference of the 'A' and 'B' inputs on each rising edge of a clock signal 'CLK' while the 'ENABLE' input is high. In order to clear the accumulated distance the asynchronous 'RST' signal must be asserted. See its detailed schematics on Appendix D.

2.6 System Controller

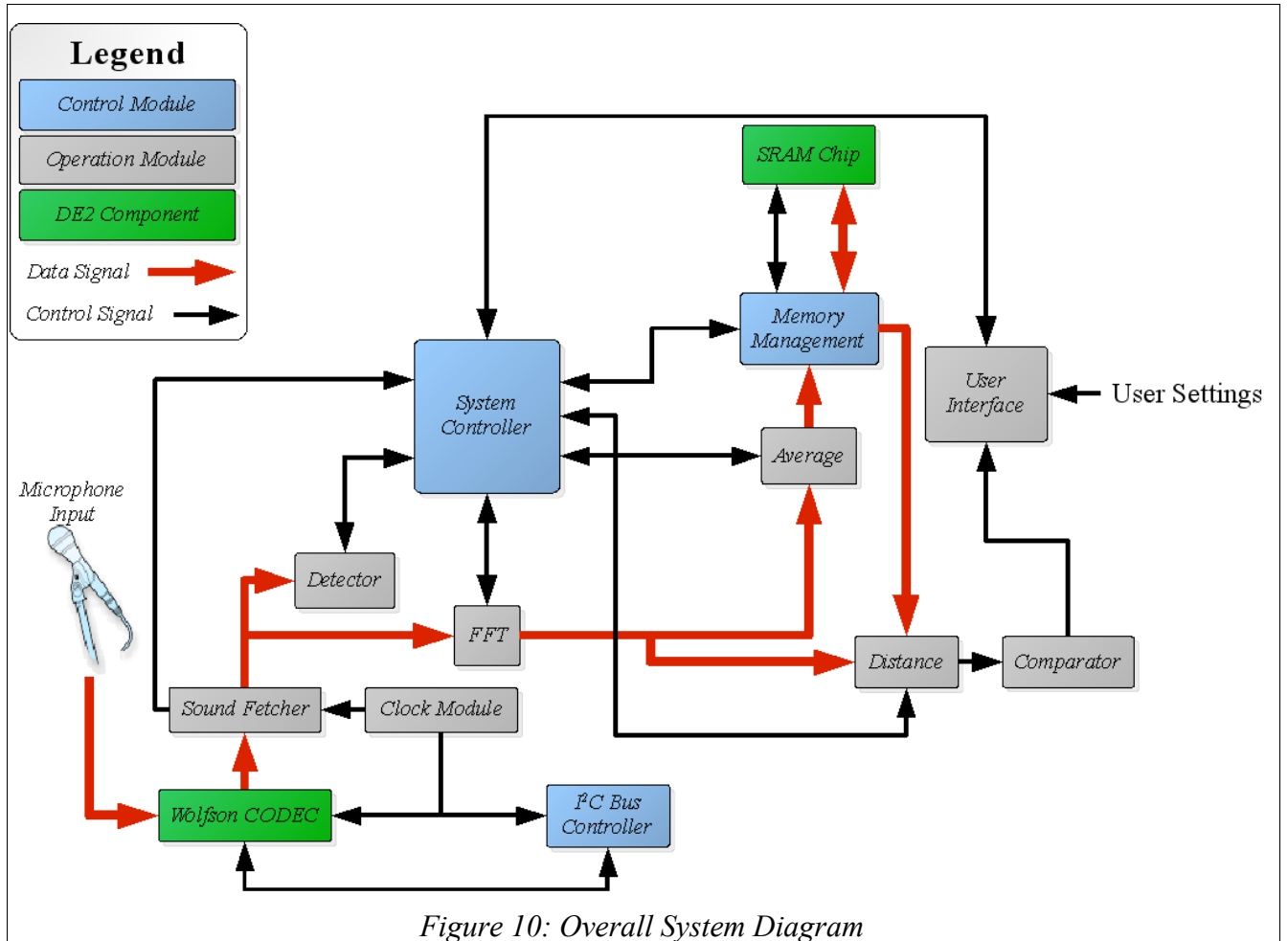


Fig. 10 shows how the modules discussed in this chapter interact with each other.

Most of the signals pass through the “System Controller” module. It controls the datapath by coordinating the modules so that the data can flow. It deals primarily with the training phase of the algorithm, since it is much more complex than the sound recognition phase. For instance, once a sound has been detected, the system controller is notified. Then, It waits for the FFT to output the data before notifying the 'Average' module it should start operating. Finally, it instructs the memory controller to store the averaged data. See the detailed user interface in Appendix E.

Chapter 3: Results

The original idea behind the project is to train the system with the word “help” several times so that, whenever the average person is in need, he or she only needs to say the word to trigger an alarm or any kind of signal. This is the reason why the results' analysis and discussion deal primarily with “help”.

3.1 FFT Issues

Many problems arise from the FFT module. First, it is difficult to acquire since it is not free. Second, it is very complex and requires investing considerable amounts of time to be fully understood before it can be used. Because of the very strict time constraints, the FFT module could not be used and the system is implemented without it. This greatly affects its accuracy, since the sound signals are compared in the time domain rather than in the frequency domain. Nevertheless, this partial implementation is functional and capable of recognizing spoken words.

3.2 MATLAB Results

Table 2 shows the results when training the system with the WAV files help2, help3, and help4.

WAV file	Distance	Verdict	Correct?	Correctness	
help1	184.9	Different	No	81.8 %	69.2 %
help2	59.3	Same	Yes		
help3	59.3	Same	Yes		
help4	118.5	Same	Yes		
help5	121.1	Same	Yes		
help6	102.5	Same	Yes		
help7	145.0	Different	Yes		
help8	108.8	Same	Yes		
help9	10.0	Same	Yes		
wu_help01	139.4	Same	Yes	0 %	
wu_help02	120.4	Same	Yes		
held01	120.5	Same	No		
held02	120.7	Same	No		

Table 2: MATLAB results for training help2, help3, and help4.

It obviously has trouble handling words that sound like “help”, such as “held” (last two rows). It nevertheless recognizes correctly “help” in about 82% of the cases.

Table 3 shows the results when training the system with the WAV files go1, go2, and go3.

WAV file	Distance	Verdict	Correct?	Correctness	
landlord01	1.7	Same	No	0 %	0 %
landlord02	1.9	Same	No		
landlord03	1.9	Same	No		
landlord04	2.4	Same	No		
landlord05	2.4	Same	No		
landlord06	3.9	Same	No		
landlord07	14.8	Same	No		
landlord08	2.0	Same	No		
landlord09	3.4	Same	No		
landlord10	2.4	Same	No		
landrover01	2.0	Same	No	0 %	0 %
landrover02	3.7	Same	No		
landmine01	2.5	Same	No	0 %	0 %
landmine02	3.0	Same	No		

Table 3: MATLAB results for training with go1, go2, and go3.

The algorithm has a lot of trouble handling small words such as “go”, since the words “landlord”, “landrover”, and “landmine” do not sound at all like “go”. This can be explained: if the word is really short, it leaves more room to noise, and thus error.

Table 4 shows the results when training the system with the WAV files nihao1, nihao2, and nihao3.

WAV file	Distance	Verdict	Correct?	Correctness	
nihao01	1.5	Same	Yes	100 %	75 %
nihao02	1.5	Same	Yes		
nihao03	1.9	Same	Yes		
nihao4	2.1	Same	Yes	0 %	

Table 4: MATLAB results with training of nihao01, nihao02, and nihao03.

In this case, the distance is extremely small, which makes the system recognize the word. In fact, the distance is also constant, because the word recordings closely resemble each other, due to the use of a headset. Note that “nihaoma” sounds a lot like “nihao”, which tricks the system (just like “held” does with “help”).

3.3 Experimental results

Word	Verdict	Correct?	Correctness	
help	Same	Yes	90.9 %	92.3 %
help	Different	No		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
help	Same	Yes		
held	Different	Yes	100 %	
held	Different	Yes		

Table 5: Experimental results for the word "help" (trained 3 times and said) by David.

Table 5 allows for a mapping of Table 2 from software to hardware. In this case, the machine is trained three times by subject “David”. The word “help” is recognized 90.9% of the time, whereas “held” is correctly ignored (100% correct) when “David” speaks. However, these percentages are respectively 45.5%, and 0% when subject “Carlos” speaks.

If during the training phase, “David” inputs two words and “Carlos” one, their percentages become respectively (when saying “help”) 72.7%, and 45.5%. When saying “held”, the machine correctly assesses that they are not saying “help” in all cases. This data was collected by saying “help” 11 times, and “held” two times.

This indicates that the training works properly, because the correctness in “David”'s results decreases, when his “participation” in the training decreases (from three times to two). On the other hand, “Carlos” correctness increases when he participates in the training.

Since the fingerprints are analyzed in the time domain, the system is much more sensible to the speed, the intonation and the surrounding noise when a word is input. Thus, the above results should be taken with caution, because the words were spoken really close to the microphone, and in a somewhat similar way each time. This can be seen in the video demonstration included in the project's CD package. Nonetheless, the results seem conclusive. Thus, despite a potential lack in accuracy, the machine is functional.

Conclusion

After applying background theory and scripting a MATLAB prototype, a speech recognition system can indeed be successfully implemented using FPGA technology.

The experimental and theoretical results show that the algorithm is accurate and fast enough for consumer product applications. Despite only partial hardware implementation due to technical difficulties, it remains functional.

Besides producing a full implementation (by including an FFT module and thus being able to analyze words in the frequency spectrum), other improvements can be done to the system. For instance, allowing the use of a variable length for the input sounds would drastically improve its performance on very short or very long words. Also, adding support for training several words would be rather simple and would increase the system flexibility.

Bibliography

Aamodt, Tor M. “A Simple Speech Recognition Algorithm for ECE341...” Toronto University. 14 June 2007 <<http://www.eecg.toronto.edu/~aamodt/ece341/speech-recognition/>>.

Boulet, Benoit. “Fundamentals of Signals & Systems”. 1st ed. Da Vinci Engineering P, 2006.

Hamblen, James. “Altera DE2 Board Resources.” Georgia Institute of Technology. 14 June 2007 <<http://users.ece.gatech.edu/~hamblen/DE2/>>.

Jones, Douglas L. “Fixed-Point Number Representation.” Connexions. 14 June 2007 <<http://cnx.org/content/m11930/latest/>>.

“PlanetMath: Discrete Cosine Transform.” PlanetMath.Org. 14 June 2007 <<http://planetmath.org/?op=getobj&from=objects&id=1469>>.

“Telephone: Answers.Com.” Answers.Com. 14 June 2007 <www.answers.com/topic/telephone>.

References

1. Altera Corp. “DE2 User Manual”. 2006. PDF Document.
2. Integrated Silicon Solution, Inc. “IS61LV25616 Data sheet”.
3. Mateos Albiach, José I. “Interfacing a processor core in FPGA to an audio system”. 2006. PDF Document. <www.diva-portal.org/diva/getDocument?urn_nbn_se_liu_diva-7191-1__fulltext.pdf>
4. Wolfson Microelectronics. “WM8731 Data sheet”. 2004. PDF Document.

Appendix A: train.m

```

=====
%
%                               One Word Training Algorithm
% Author:
%   Carlos Asmat                260148251
%   David Lopez Sanzo          260146414
%   Kanwen Wu                  260045745
%
% Name: train.m
%
% Description: This is the first part of the entire voice recognition
% process. This script is produces the "fingerprint" of a sound by
% averaging many (as many as the user wants) versions of it.
% The "fingerprint" will be used in the second stage of the voice
% recognition algorithm.
%
% Note: the algorithm fuctions properly if the input wav sound sampling
% frequency is 22050.
%
%
% Creattion: May 6 2007
% Last Modification: May 30 2007
=====

% Clear old graphs and command history
clear all
close all
close all hidden
clc

=====
% Variables and parameters definition
=====

% Quantization parameters
F = 4;      % Downsampling factor
qBits = 8;  % Number of bits for quantization
step = 2*(2^(-qBits)); % Quantization step

% "Begining of Sound" Detector parameters
win = 1000; % Window length
l = 1.024;  % Sound length set to 1.024s
th = 0.05;  % Begining of sound threshold

% FFT parameters
N = 1024;   % Number of point for each FFT
n = 5;      % Nmber of FFTs

X = zeros(n,N); % FFT vector initialisation
FingerPrint = zeros(n,N); % Fingerprint vector initialisation

=====
% Training Algorithm
=====

% Ask how many times the word should be trained.

```

```

times = int32(input('How many times do you want to train? '));

for tim = 1:times % The training algorithm is repeated 'times' times.

    % Prompted the user for a sound file until a valid filename is received.
    % The sound is stored as a column vector in 'x'.
    bob = 1;
    while(bob == 1)
        str = strcat('Enter the file name for the training sound (', int2str(tim),
        '/', int2str(times), ') : ');
        fileName = input(str, 's');
        bob = 0;

        try
            [x, sf, nbits] = wavread(fileName);
        catch
            disp(['[] Error: Invalid file name.']);
            bob = 1;
        end
    end

    % Decimate the sound by factor F.
    xd = decimate(x,F);

    % Quantize the sound.

    [I, xq] = quantiz(xd, -1:step:1-step, -1:step:1);

    %=====
    % Detect the beginning of the word
    %=====

    ptr = 1; % Initialize pointer.

    ave1 = mean(xq(ptr:ptr+win)); % Initialization of average windows.
    ave2 = ave1;

    % Go through the sound until the difference between the average of two
    % adjacent windows is significant.
    while abs(ave1-ave2) < th
        ptr = ptr + win;
        ave2 = ave1;
        ave1 = mean(abs(xq(ptr:ptr+win)));
    end
    s = xq(ptr:int32(ptr+1*sf/F)); % Store the detected sound in 's'.

    % Compute 'n' 'N' points FFT of s and store it in 'X'.
    for k = 1:n
        X(k, 1:N) = real(fft(s((k-1)*N+1):k*N), N);
    end

    FingerPrint = FingerPrint + X; % Add the FFTs for each trial together.
end

FingerPrint = FingerPrint/double(times); % Average the FFTs.

% Display the fingerprint.
% Note: only half of the fft is displayed since the fft of a real signal
% is half redundant.

```

```
figure('Name','Word Fingerprint','NumberTitle','off');
for k = 1:n
    subplot(n,1,k);
    plot(FingerPrint(k, 1:N/2));
    xlim([0 N/2]);
    if k == int32(n/2)
        ylabel ('Amplitude');
    end
end
xlabel ('\omega \times N \div 4\pi');

% Save data for the recognition phase of the algorithm.
save('Training_Data', 'FingerPrint', 'n', 'N', 'F', 'qBits', 'step', 'win', 'l',
'th');
```

Appendix B: recogniz.m

```

=====
%
%                               One Word Recognizer
% Author:
%   Carlos Asmat                260148251
%   David Lopez Sanzo          260146414
%   Kanwen Wu                  260045745
%
% Name: recogniz.m
%
% Description: This is the second part of the entire voice recognition
% process. This script is compares the "fingerprint" of a sound to the
% reference fingerprint created in the training phase.
%
% Note: the algorithm fuctions properly if the input wav sound sampling
% frequency is 22050.
%
%
% Creattion: May 6 2007
% Last Modification: May 30 2007
=====

% Clear old graphs and command history
clear all
close all
close all hidden
clc

=====
% Variables and parameters definition
=====

% Load data from the training phase
try
    load('Training_Data');
catch
    disp '[:] Error: Please run train.m first.';
end

maxDis = 140; % Maximum acceptable distance.
dis = 0; % Initialize the distance variable.
X = zeros(n,N); % FFT vector initialisation
Distance = zeros(n,1);
% Prompted the user for a sound file untill a valid filename is received.
% The sound is atored as a column vector in 'x'.
bob = 1;
while(bob == 1)
    fileName = input('Enter the file name for the sound to be recognized: ', 's');
    bob = 0;

    try
        [x, sf, nbits] = wavread(fileName);
    catch
        disp '[:] Error: Invalid file name.';
        bob = 1;
    end
end

```

```

end

% Decimate the signal by factor F.
xd = decimate(x,F);

% Quantize the signal.
[I, xq] = quantiz(xd, -1:step:1-step, -1:step:1);

%=====
% Detect the beginning of the word
%=====
ptr = 1; % Initialize pointer.

ave1 = mean(xq(ptr:ptr+win)); % Initialization of average windows.
ave2 = ave1;

% Go through the sound untill the difference between the average of two
% adjacent windows is significant.
try
    while abs(ave1-ave2)<th
        ptr = ptr + win;
        ave2 = ave1;
        ave1 = mean(abs(xq(ptr:ptr+win)));
    end
catch
    ptr = 200;
end

s = xq(ptr:int32(ptr+1*sf/F));

% Compute 'n' 'N' points FFT of s and store it in 'X'.
for k = 1:n
    X(k, 1:N) = real(fft(s((k-1)*N+1):k*N), N);
end

% Compute the distance between 'X' and 'FingerPrint'.
dis = X - FingerPrint;
for k = 1:n
    Distance(k) = sqrt(dis(k, 1:N)*dis(k, 1:N)');
end

Distance = mean(Distance)
if Distance > maxDis
    disp 'Word not recognized';
else
    disp 'Word recognized!';
end

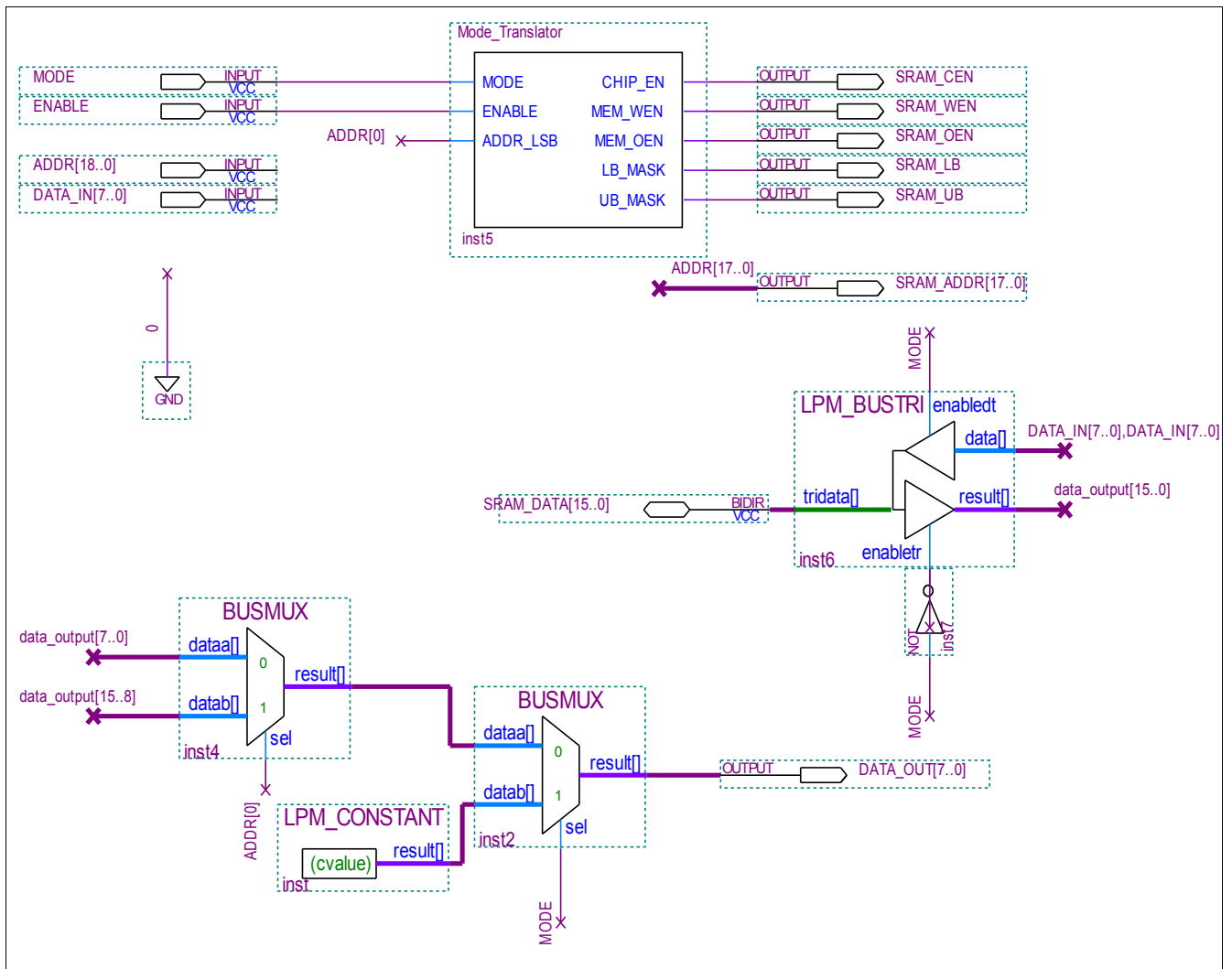
% Display the reference fingerprint.
% Note: only half of the fft is displayed since the fft of a real signal
% is half redundant.
figure('Name','Reference Fingerprint','NumberTitle','off');
for k = 1:n
    subplot(n,1,k);
    plot(FingerPrint(k, 1:N/2));
    xlim([0 N/2]);
    if k == int32(n/2)
        ylabel ('Amplitude');
    end
end

```

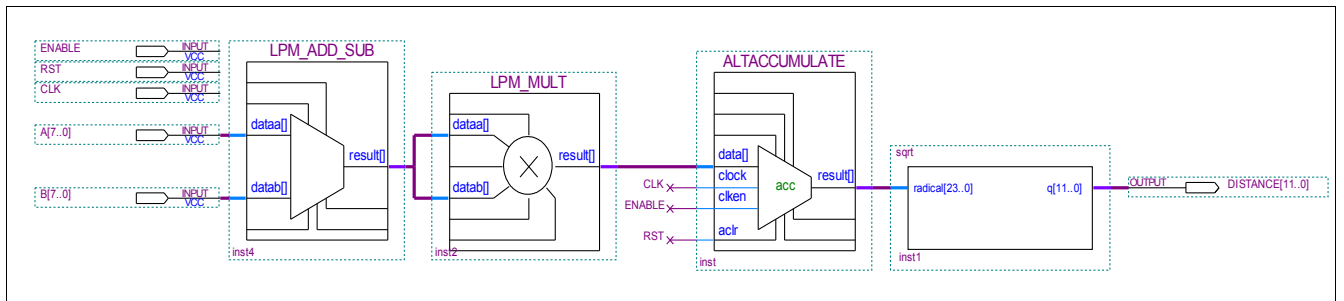
```
end
xlabel ('\omega \times N \div 4\pi');

% Display the word fingerprint.
% Note: only half of the fft is displayed since the fft of a real signal
% is half redundant.
figure('Name','Word Fingerprint','NumberTitle','off');
for k = 1:n
    subplot(n,1,k);
    plot(X(k, 1:N/2));
    xlim([0 N/2]);
    if k == int32(n/2)
        ylabel ('Amplitude');
    end
end
xlabel ('\omega \times N \div 4\pi');
```

Appendix C: Schematics for the Memory Controller



Appendix D: Schematics for the Distance Module



Appendix E: Detailed User Interface

