HARDWARE IMPLEMENTATION OF A LOG-VITERBI DECODER FOR HMM BASED ISOLATED SPEECH RECOGNITION

A Project Report

submitted by

N VENKATA NAVEEN

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **HARDWARE IMPLEMENTATION OF A LOG-VITERBI DECODER FOR HMM BASED ISOLATED SPEECH RECOGNITION**, submitted by **N VENKATA NAVEEN**, to the Indian Institute of Technology, Madras, for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Janakiraman Viraraghavan
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 24th May 2018

ACKNOWLEDGEMENTS

I have put my best efforts into completion of this project. This wouldn't have been possible without the support of many individuals and organization. I would like to thank my project guide Prof.Janakiraman Veeraraghavan for choosing me and giving me a project that suits my interests and goals. He has helped me with his precious knowledge in the fields of Digital IC Design, Speech Processing and Hardware design at each and every stage of the project. He has always been supportive and open for discussing new ideas and helped me get through many of my mistakes.

I am also fortunate to have the company of Prof.Nitin Chandrachoodan for his invaluable inputs in the RTL design. His vast experience in the field of FPGA/RTL design has helped me deal with the many errors that were present in the initial design phase.

Also a special mention to Mr.Venkat Raghavan for providing the MATLAB source code and helping me in get the code running. Without this code this project wouldn't have been possible.

I would also like to thank Janaki Madam for helping me with the setup of software/tools in the lab. Finally, I would thank my lab mate Prithvi Raj who has helped me with the understanding of basics principles of Markov Models which eased my process of going through the research papers.My knowledge on many of the topics presented in the thesis was limited at the beginning of the project, but through the course of the academic year I was able to appreciate the concepts thoroughly.

ABSTRACT

KEYWORDS: Hidden Markov Model ; Viterbi; FPGA; Trellis, Verilog.

The thesis presents a hardware implementation(based on fixed point implementation) of log-viterbi decoder to recognize isolated speech consisting of set of words based on the concepts of Hidden Markov Models(HMM). The circuit designed predicts the word spoken based on the probabilities obtained from the trained sets of pre-calculated data.

A sample case of isolated spoken digits consisting of words from zero to nine has been analyzed for various possible utterances. A confusion matrix shown at the end provides an estimate of the performance of the system with respect to accuracy. The pre-processing steps required prior to decoding stage and all the HMM-data required has been generated using the MATLAB source code.Therefore all the computations done are based on the data that has been generated from this code and hence the simulation results obtained have been verified against the MATLAB's outputs.

The architecture design and style of RTL coding used with regards to parameterization so as to reuse the code for future purposes easily has been presented. The benefits of using the fixed point implementation with an optimized register bit-lengths at each stage compared to floating point computations(as done in MATLAB) have been discussed. Further, the synthesis and implementation results targetting an FPGA have also been presented and analysed.

The work provides an useful insight into tackling the bigger problem of designing an optimized speaker independent sentence recognition system on hardware.

Contents

ACKNOWLEDGEMENTS **i**

ABSTRACT **ii**

LIST OF TABLES **v**

LIST OF FIGURES **vi**

ABBREVIATIONS **vii**

NOTATION **viii**

1 INTRODUCTION **1**

2 ISOLATED SPEECH RECOGNITION **3**

2.1 Pre-processing steps 3

2.2 Speech Modelling using HMMs 4

2.3 Viterbi-Decoder 7

2.3.1 Viterbi Algorithm-Theory and Equations 7

2.3.2 Trellis Diagrams 7

2.4 Predicting the word 8

3 HARDWARE ARCHITECTURE AND IMPLEMENTATION **10**

3.1 Log-Viterbi Decoder 10

3.2 Hidden Markov Models for Isolated Digits 11

3.3 Register Sizes For Fixed Point Arithmetic 13

3.3.1 Choosing '∞' 13

3.4 RTL Hierarchy 14

3.5 Block Diagram of Entire System 15

4 RESULTS- SIMULATION AND SYNTHESIS **16**

4.1 Confusion Matrix 17

4.2 Synthesis 17

5 CONCLUSIONS AND FUTURE WORK **20**

6 APPENDIX- REFERENCES **21**

List of Tables

4.1 Table showing the confusion matrix generated by 150 different simulations, fifteen per each word 18

List of Figures

2.1 Block diagram of an Isolated Speech Recognition system. 3

2.2 Pre-processing stages before the Decoder 4

2.3 Example of an Hidden Markov Model 5

2.4 HMM with a transition from higher numbered state to lower . . 6

3.1 HMM of an Isolated Digit 11

3.2 Block diagram of a Viterbi_d Module 14

3.3 Block diagram for mapping input symbols for each state 15

3.4 Block diagram of the entire decoding hardware 15

4.1 Wave forms for a correctly recognized digit 16

4.2 Wave forms for a wrongly recognized digit 17

4.3 Figure shows the number of adders of various bit-lenghts used . 18

4.4 Figure showing the total number of different types of MUX used 19

4.5 Figure showing percentage utilization of hardware on Zybo board 19

ABBREVIATIONS

HMM	Hidden Markov Model
FPGA	Field-Programmable Gate Array
RTL	Register Transfer Logic
ASIC	Application Specific Integrated Circuit

NOTATION

δ	Cost metric associated to a state in an HMM
α_{ij}	Transition probability of a given HMM upon going from state i to j
b	Symbol probability of a given state for a given observation
π	Possible list of state transition probabilities at time T=0

Chapter 1

INTRODUCTION

Real time speech recognition is of high importance in our day to day lives. It is being used in our mobile phones and laptops for various purposes like virtual assistance that performs useful tasks like searching and browsing. Apart from this speech recognition also plays an integral role in helping those who are suffering from motor speech disorders.

Plenty of speech recognition devices have been emerging but most of them still lack high accuracy and are also slow. As there are many ways a word can be pronounced/uttered, the main challenge of a speech recognition system is to accurately recognize them and be speaker independent. HMM based recognition system provide much better accuracy and can be made fast by designing appropriately. In view of this, lots of prior training of HMM models and faster recognition algorithms needs to be implemented. To perform such computing intensive tasks software based solutions tend to be slow. Hardware acceleration would be a better approach as a dedicated hardware would perform the particular task faster due to parallel computing capability and lesser overheads.

The ultimate goal of designing a dedicated hardware for speech recognition would be to accurately recognize the sentence spoken as fast as possible and occupy as much less area as possible. A sentence recognition system would be based on continuous HMMs and are very difficult to perform calculation on hardware. Hence a better beginning to solving such complex problems would be to design an isolated word recognition system. Since an isolated word stops after the completion of utterance, we can model them as discrete HMMs which has simpler equations to work with.

The organization of the thesis from the next chapter is as follows. Chapter 2 presents basics of an isolated speech recognition system, its main components and also introduces to the basics of Hidden Markov Models(HMMs) following which the equations required for the viterbi decoder have been analyzed. Chapter

3 presents the details of the actual implementation of the architecture for Log-viterbi decoder. Chapter 4 presents the simulation ans synthesis results obtained for the logic designed. Chapter 5 includes conclusions and future work.

Chapter 2

ISOLATED SPEECH RECOGNITION

Isolated speech recognition consists of predicting the spoken word from a given set of words. The spoken words will be transformed to an equivalent form of a sequence of observational states which are obtained through the pre-processing stage and will be sent to a decoding stage where we predict the best suitable word based on the given HMM models as shown in Fig. 2.1. As we build the recognition system for only a set of words, if the input word spoken is different from this set, we get a wrong output(one among the set).

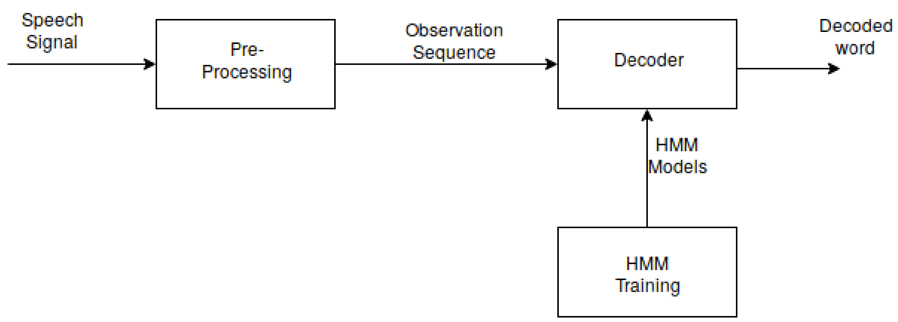


Figure 2.1: Block diagram of an Isolated Speech Recognition system.

2.1 Pre-processing steps

Once the speech signal is received it undergoes a lot of steps before being sent to the decoder. Brief description of these steps is provided below.

- **Pre-emphasis:** This step is to boost certain frequencies that are highly susceptible to noise and spectrally flatten the signal.
- **Framing:** The digitized speech signal is divided into various overlapping frames of equal size(typically 10-20ms).
- **Windowing:**Windowing is done to minimize the effects of framing which might disrupt the continuity.

- **MFCC Feature extraction:** This step involves the conversion of the sampled signal frames into feature extracted vectors. The steps involved are 1.Taking FFT of the signal 2.Applying Mel-triangular filters and converting to the log-power spectrum 3.Taking Discrete Cosine Transform(DCT).
- **Clustering:** Since we are dealing with the isolated speech involving discrete HMMs, we can actually cluster the obtained featured vectors into various centroids thus allocating each MFCC vector a centroid and hence the input to the decoder is an observations sequence consisting of these centroids.

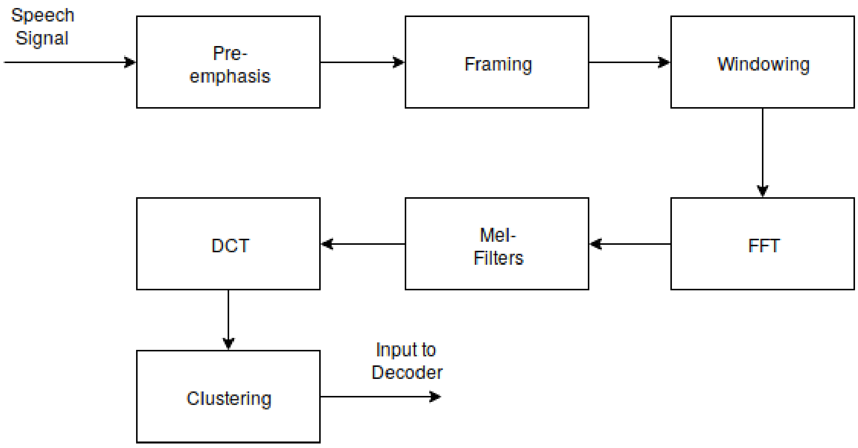


Figure 2.2: Pre-processing stages before the Decoder

2.2 Speech Modelling using HMMs

The Hidden Markov Model(HMM) is a statistical model in which observations are functions of states in terms of probability. In Hidden Markov Models(HMM), the actual states are unknown(hidden) but their state transition probabilities are known. Based on the observation sequence one can predict the best possible sequence of states that could have resulted in such observations. An HMM($\boldsymbol{\lambda}$) can be modelled by three parameters namely **A, B** and $\boldsymbol{\pi}$**.**

$$\boldsymbol{\lambda} = (\mathbf{A},\ \mathbf{B}, \boldsymbol{\pi})$$

Where **A** is the state transition matrix. It show all the state transition probabilities. **B** represents the list of all possible observations from each state and also their probabilities. $\boldsymbol{\pi}$ represent the initial state of the system and contains the probabilities of reaching each state at time t=0. An example of an Hidden Markov Model is shown below in Fig. 2.2

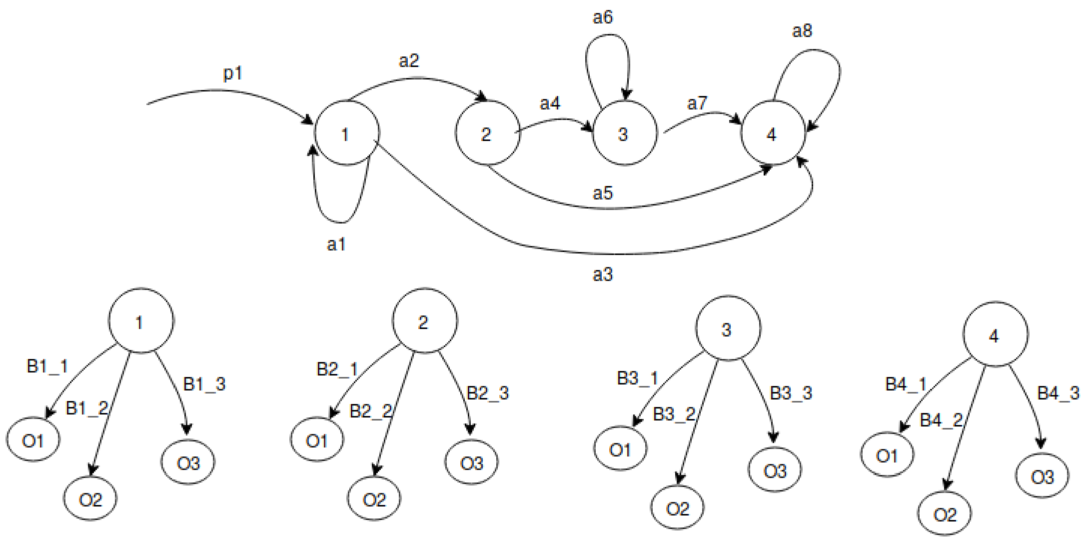


Figure 2.3: Example of an Hidden Markov Model

For this example the states are numbered as 1,2,3 and 4. The matrices **A**, $\boldsymbol{\pi}$, **B1**, **B2**, **B3** and **B4** are as follows.

$$\mathbf{A}= \begin{bmatrix} \boldsymbol{a1} & \boldsymbol{a2} & \mathbf{0} & \boldsymbol{a3} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{a4} & \boldsymbol{a5} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{a6} & \boldsymbol{a7} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{a8} \end{bmatrix}$$

$$\pi = \begin{bmatrix} \boldsymbol{p1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{B1}= \begin{bmatrix} \boldsymbol{B1\_1} & \boldsymbol{B1\_2} & \boldsymbol{B1\_3} \end{bmatrix} \mathbf{B2}= \begin{bmatrix} \boldsymbol{B2\_1} & \boldsymbol{B2\_2} & \boldsymbol{B2\_3} \end{bmatrix}$$

$$\mathbf{B3}= \begin{bmatrix} \boldsymbol{B3\_1} & \boldsymbol{B3\_2} & \boldsymbol{B3\_3} \end{bmatrix} \mathbf{B4}= \begin{bmatrix} \boldsymbol{B4\_1} & \boldsymbol{B4\_2} & \boldsymbol{B4\_3} \end{bmatrix}$$

Rabiner's paper on Hidden Markov Models describes three problem statements that are possible relating to these HMMs. A brief discussion on these problems and how to solve them is discussed below.

- **Evaluation**: It involves calculation of the probability of observing a sequence $(\boldsymbol{O_1, O_2.., O_T})$ given the module $\boldsymbol{\lambda}$ for every possible state sequence. Assuming a possible state sequence is $\boldsymbol{Q_1 = q_1, q_2, q_3, ..q_t}$ we can write the probability of observing he sequence **'O'** such that the state transitions are as in $'Q'_1$ as

$$P(O, Q_1|\lambda) = (\pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} ... a_{q_{T-1} q_T})(b_{q_1}(O_1).b_{q_2}(O_2)...b_{q_T}(O_T)) \tag{2.1}$$

$$P(O|\lambda) = \sum_k (P(O, Q_k|\lambda)) \tag{2.2}$$

This way of calculating the probability normally would involve $2TN^T$ multiplications which means exponential dependency on time 'T'. Instead we can use clever algorithms like **Forward Algorithm(** TN^2) to avoid re-multiplications at each stage.

- **Decoding**: This process involves in predicting the most probable path $Q_x = q_1, q_2, q_3, ..q_t$ that could resulted in a given observation sequence $(O_1, O_2.., O_T)$ given the HMM λ. This is the problem that will be of interest in speech recognition as it can be translated to finding the sequence of states(Phonemes) that could have resulted in the observed sequence(Quantized Vectors of speech) and hence we could predict the word spoken.

- **Learning**: This process involves in actual training of the HMM model i.e., adjusting the **A, B,** π parameters to maximize the probability of the observed sequence.

In the context of speech, each HMM model will correspond to a word and these models are developed through training. For a phoneme based HMM model each of the state correspond to phoneme transition or each phoneme itself corresponds to 2 or more states. Also since speech signal moves forward with time, there shouldn't be as such any transitions from a higher numbered state to a lower numbered state. For example the following HMM in Fig 2.3 is not possible in a speech scenario.

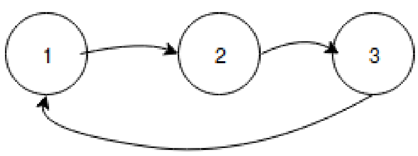


Figure 2.4: HMM with a transition from higher numbered state to lower

2.3 Viterbi-Decoder

Once the pre-processing steps and the model training of the word set is done, the next important step is the decoder. The decoder block is the one which actually estimates the most probable word that was spoken. Since we can have various utterances for each word and noise added to signals, the observational sequence which comes as input to the decoder varies from the trained model for that word. So the decoding algorithm must be robust enough to overcome this difficulty. The most widely used decoding algorithm is the viterbi algorithm because of its ability to largely avoid the re-computations at each step.

2.3.1 Viterbi Algorithm-Theory and Equations

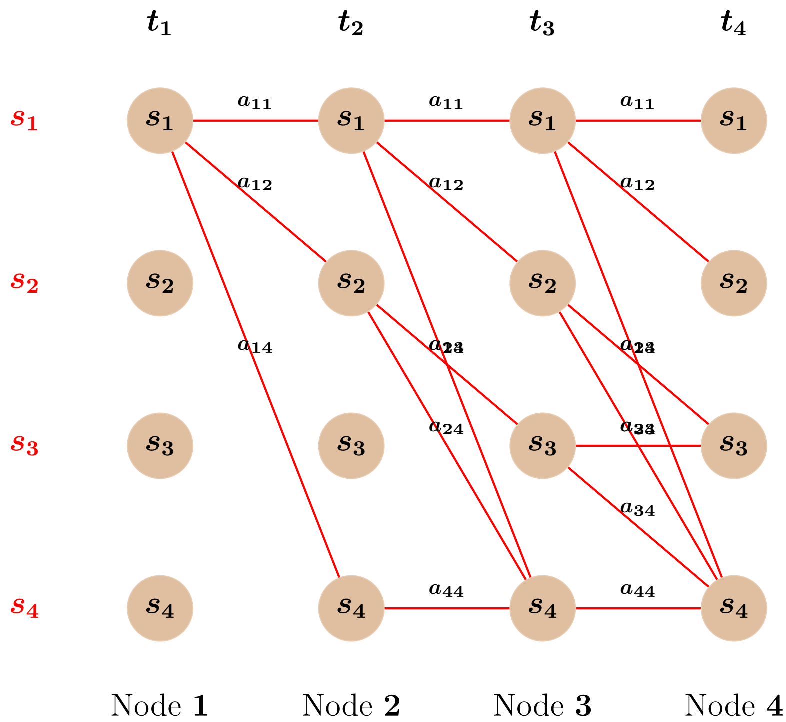
As discussed above the inputs to the viterbi decoder would be the observation sequence which will be entering sequentially from time t=0 till t=n. The HMM trained data will be available to this decoder at each of these time instances. Suppose each of the trained HMM model has 'k' distinct states, we assign a metric($\boldsymbol{\delta(i)}$) to the ith state and hence 'k' such metrics. The term $\boldsymbol{\delta_t(i)}$ indicates the metric of state 'i' at time T=t. The notations for **A, B,** $\boldsymbol{\pi}$ matrices are same as discussed above with an element in **A**, a$_{ij}$ indicating transition probability from state 'i' to state 'j'. Each element of **B** corresponding to a state 'j' and input observation $\boldsymbol{O_k}$ at time T=t is represented by $\boldsymbol{b_j(O_t)}$ and each element of $\boldsymbol{\pi}$ corresponding to state 'i' is represented by $\boldsymbol{\pi_i}$. Using these notations the equations for updating the metrics for each state are as follows.

$$\boldsymbol{\delta_1(i) = \pi_i * b_i(O_1); \quad t = 1, 1 \leq i \leq k} \tag{2.3}$$

$$\boldsymbol{\delta_t(j) = \max_{i=1,2\ldots k}(\delta_{t-1}(i) * a_{ij}) * b_j(O_t); \quad 2 \leq t \leq T, 1 \leq j \leq k} \tag{2.4}$$

2.3.2 Trellis Diagrams

Trellis diagrams provide an easier view of the flow of the state transitions at each instance of time. They are also helpful in tracing back the path(sequence of states) that could have most probably led to the observation sequence. Each column of the trellis diagram indicates a time instance and also shows the input observation at that time instance. Each column also contains all the states that are possible for the given model. The arrows from time to other shows all the possible transitions possible. The trellis diagram for the HMM model shown in Fig. 2.3 is as shown below.



As we can see in the trellis diagram, for time T=$\boldsymbol{t_1}$, the probability for the input observation is searched only for state $\boldsymbol{s_1}$ since $\boldsymbol{\pi_i}$=0 for other states. Hence metric for only the first state is updated. At time T=$\boldsymbol{t_2}$, the state $\boldsymbol{s_3}$ cannot be reached and hence metrics for the states $\boldsymbol{s_1, s_2, s_3}$ will only be updated according to the formula (2.2)

2.4 Predicting the word

Once we reach the end of the input sequence after time T=$\boldsymbol{t_n}$, we get the updated metrics for each state. We now calculate the probability of the occurrence of the input sequence($\boldsymbol{O_t}$) given the model $\boldsymbol{\lambda_x}$ for each word as

$$\boldsymbol{P_x(O|\lambda_x) = \max_{1\leq i\leq k}(\delta_T(i)); \quad 1 \leq x \leq cardinality(word-set)} \tag{2.5}$$

And the word that could have been spoken is the $\boldsymbol{y^{th}}$ word in the word-set such that.

$$\boldsymbol{y} | \max_{\boldsymbol{1 \leq x \leq cardinality(word-set)}} (\boldsymbol{P_x}) = \boldsymbol{P_y} \tag{2.6}$$

Chapter 3

HARDWARE ARCHITECTURE AND IMPLEMENTATION

In the second chapter the procedure for predicting the word for the given speech signal has been described along with the necessary equations. To implement this in hardware it is important to describe the architecture and then the data flow in terms of logic circuits. One must also keep in mind the estimate of Area, Power and Speed of the circuit. The reason is that, even though we are designing the system in hardware primarily to speed up the process, we may end up spending a lot of power and area which can make the design useless. Hence, we might actually have to re-construct the equations themselves for the betterment of the design. One such idea is to use Log-Viterbi decoder.

3.1 Log-Viterbi Decoder

A Log-Viterbi decoder is same as the Viterbi decoder except that we take logarithm of each of the equations 2.1, 2.2 and 2.3. We can re-write these equations as follows.

$$\boldsymbol{log(\delta_1(i)) = log(\pi_i) + log(b_i(O_1)); \quad t = 1, 1 \leq i \leq k} \tag{3.1}$$

$$\boldsymbol{log(\delta_t(j)) = \max_{i=1,2\ldots k}(log(\delta_{t-1}(i)) + log(a_{ij})) + log(b_j(O_t)); \quad 2 \leq t \leq T, 1 \leq j \leq k} \tag{3.2}$$

$$\boldsymbol{P_x(O|\lambda_x) = \max_{1 \leq i \leq k}(log(\delta_T(i))); \quad 1 \leq x \leq cardinality(word-set)} \tag{3.3}$$

As we can see, the difference in both set of equations is that we have been able to replace multiplications with additions. This is very useful in the context of hardware because a multiplier would usually take more area and also is slower than an adder. But one can see that it has resulted in need of calculating logarithms at each step which itself is area and time taking process on hardware. If we observe the equations carefully we can actually pre-compute all the logarithms and store them in memory. Specifically in equation 3.1 $\boldsymbol{\pi_i}$ and $\boldsymbol{b_i(O_1)}$ belongs the trained HMM data since we have all of this data beforehand we can compute the logarithms before the decoder stage itself. Similarly in equation 3.2 $\boldsymbol{a_{ij}}$ are the state transitions of the word models and are all available prior to decoding stage.

3.2 Hidden Markov Models for Isolated Digits

The word-set for which the decoder has been built consists of digits(zero-nine). The trained model generation and speech pre-processing has been done on Matlab and some of the properties of the generated data from the Matlab are as follows.

- Each word is represented by a set of seven states and the transitions among these states are as shown below in Fig 3.1.

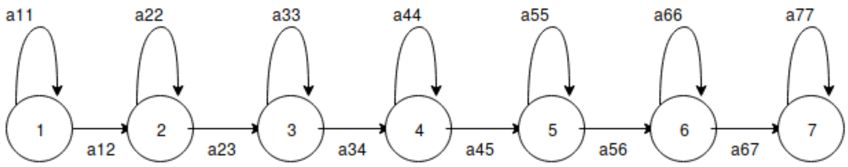
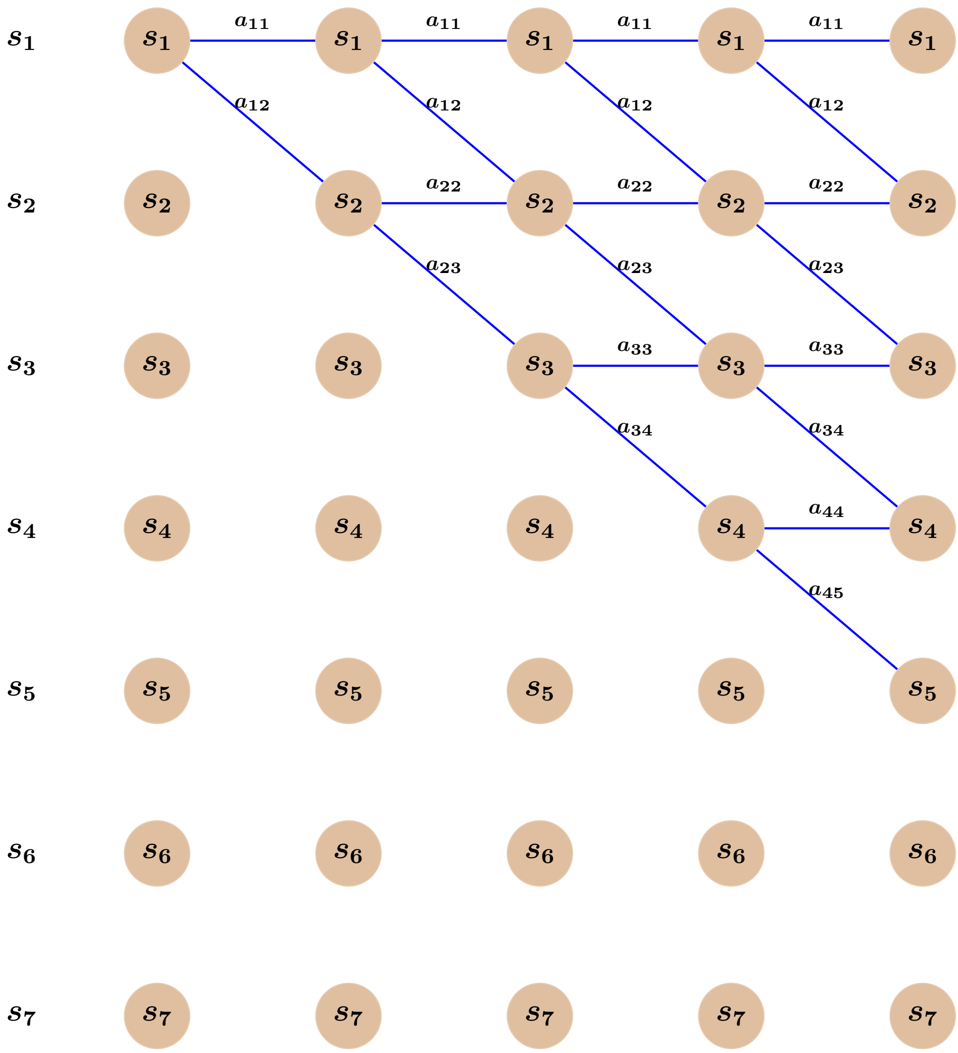


Figure 3.1: HMM of an Isolated Digit

- The trellis diagram for the HMM shown in Fig 3.1 would be as shown below. In the context of Hardware the times $\boldsymbol{t_1, t_2..}$ indicate the rising edges of clock during which the updating of the metrics will be done according to equations 3.1 and 3.2



- From the Fig 3.1 we can see that there are 13 transition probabilities $(\boldsymbol{a_{11}, a_{12}, ..., a_{77}})$ and hence we need 13 registers to store these numbers per word. Generalizing for higher number of states with similar state transition property we need **(2N-1)** registers for **N** states per word. Hence for a ten word system we need 10***(2N-1)** registers

- There are 64 different possible observations for each state in a given HMM. Hence there would be 64*7 probabilities per word and 64*7*10 for the entire system. Hence we need an additional 4480 registers.

- For all the words in the system the initial state probabilities are as follows

$$\boldsymbol{\pi_i} = \begin{cases} \mathbf{1}, & \text{if } \boldsymbol{i = 1} \\ \mathbf{0}, & \text{otherwise} \end{cases} \tag{3.4}$$

  Which means that at time T=0 only the first state is reached. Since this is fixed we don't need to remember(store) any of the probabilities from the $\boldsymbol{\pi}$ matrix.

- A word has various utterances and each such utterance gives different observation sequences of variable length. Hence it is necessary to generate a signal(Flag) to indicate the end of the input sequence.

3.3 Register Sizes For Fixed Point Arithmetic

Register bit-widths depends on factors like maximum and minimum values of the probabilities in trained models, and also the least and largest possible values inside a particular module. The tricky part is that since we are storing the logarithmic values and not the actual probabilities, we must account for the signs of the numbers and in particular a reasonable form to represent '$-\infty$'. As a result of appearance of negative numbers each of the logarithmic values will be stored in $\boldsymbol{2's}$ complement form and all the operations performed will also be based on $\boldsymbol{2's}$ complement arithmetic.

3.3.1 Choosing '∞'

All the probabilities are non-negative quantities which will result in all their logarithmic values becoming negative. Hence adding these negative quantities according to equations 3.1 and 3.2 would lead to larger negative values. Hence our choice of infinity should be as negative as possible. In a counter argument we see that if we have a case where the most of the probabilities are closer to one(log values closer to zero) we can afford to choose smaller bit widths for the register and hence a relatively higher value for '$-\infty$'. In this case we must choose the value carefully so as to avoid approaching '$-\infty$' values during successive logarithmic additions.

Finally based on all the above observations, each of the transition probabilities and the output observation probabilities have been assigned 16 bits among which 6 bits corresponding to integer part and 10 bits corresponding to decimal part(6.10 fixed point format). Where as the metric registers corresponding to each state has been assigned 16+logk number of bits where 'k' is the maximum number of the length of the input sequence. This is because each incoming input would result in at most one addition of two 16 bit numbers.

3.4 RTL Hierarchy

Based on the bit-widths for registers obtained above, the RTL(Verilog) has been designed as follows. There is a module $\boldsymbol{Viterbi\_d}$ which basically performs the operations described in equations 3.1 and 3.2 for a single word. At each positive edge of the clock cycle the metric registers $\boldsymbol{\delta_i(j)}$ are updated. Also these metric register's outputs are connected to a series of comparators so as to output the maximum among all the metrics. the The circuit block diagram for such a process is as shown in Fig 3.2. As we can see from the figure that the inputs to this

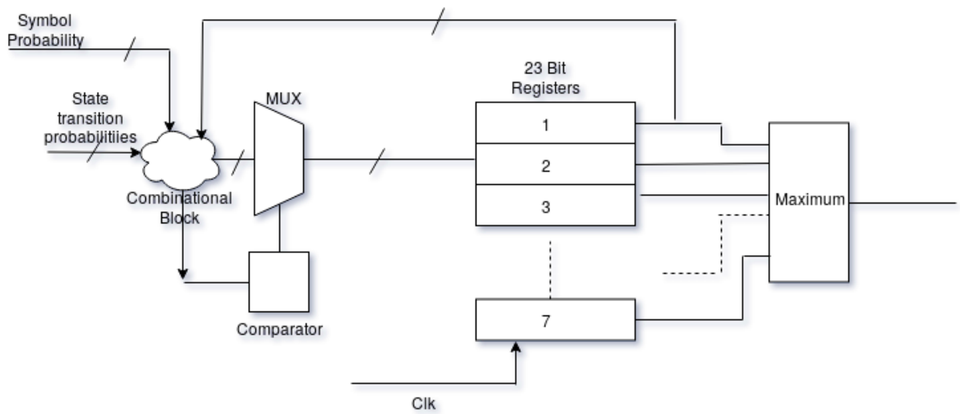


Figure 3.2: Block diagram of a Viterbi_d Module

module are symbol probabilities which change for every clock cycle and state transition probabilities which are constant values and are directly mapped from memory. The symbol probabilities are transferred from the registers through a multiplexer which has the select line as the observation symbol as shown in Fig 3.3. Since there are seven states per word and there are ten words, we will have 70 such multiplexers.Now that the entire description of viterbi_d module is done, we can build the entire decoding system by instantiating this module for 10 times and passing their output to another set of comparators to find out the maximum probability among the 10 values. This constitutes the 'top' module and is the complete circuit that was supposed to be built. It takes in CLK and observation symbol as its inputs and gives out the predicted word.

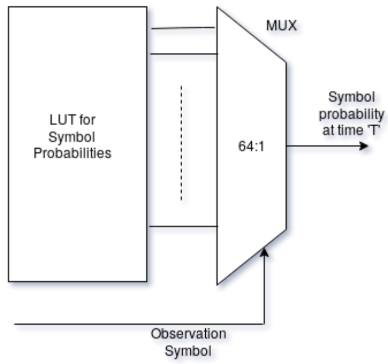


Figure 3.3: Block diagram for mapping input symbols for each state

3.5 Block Diagram of Entire System

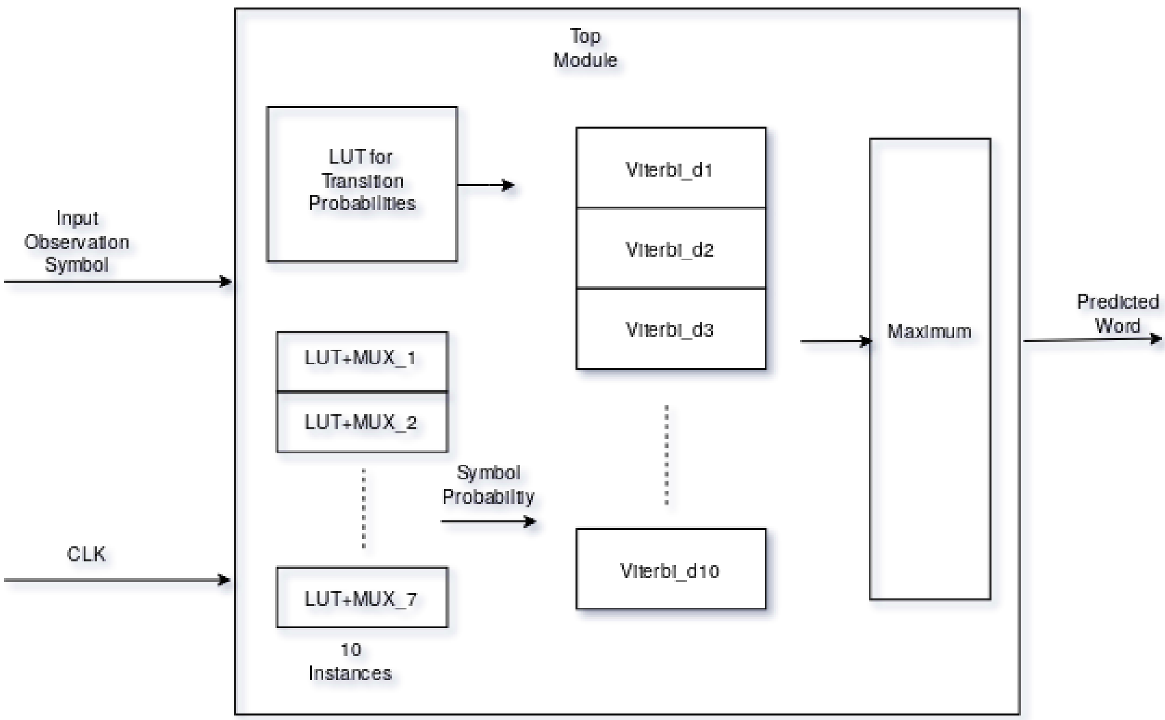


Figure 3.4: Block diagram of the entire decoding hardware

Chapter 4

RESULTS- SIMULATION AND SYNTHESIS

Based on the RTL designed using the architecture described in Chapter 3, various simulation and synthesis results have been analyzed. The simulation tool used was 'iverilog' and 'gtkwave' has been used to view the waveforms. The input sequences to the top module have been generated in the MATLAB. A total of 112 possible utterances for each digit have been provided in the MATLAB. Consider the following result Fig 4.1 that was obtained when the system was fed with a sequence of inputs corresponding to digit 'six'

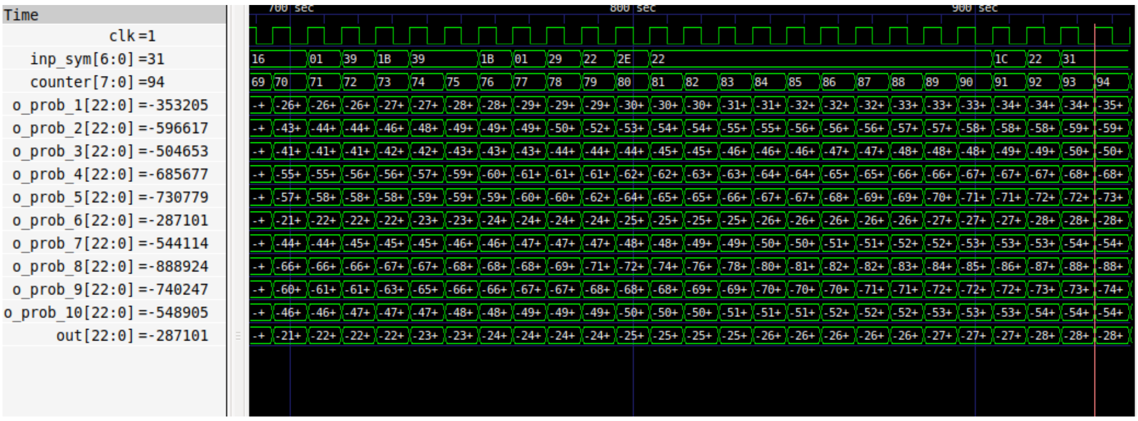


Figure 4.1: Wave forms for a correctly recognized digit

There are 14 different signals that are being projected here. The first signal being the clock keeps running throughout changing its value from 0 to 1. The second signal is the input symbol to the top module and third signal indicates the number of inputs that have entered the systems. From the figure it is clear that the sequence has a total of 94 inputs. The signals from four to thirteen indicate the outputs of each of the 10 ***viterbi_d*** modules. The fourteenth signal is the maximum probability among the ten outputs of ***viterbi_d*** module. As we can see the output corresponding to sixth signal is the highest indicating that the word is correctly detected.

Also consider the figure 4.2 whose inputs(a total of 106 symbols) actually correspond to digit nine. But the digit showing the highest probability is 'one' with 'nine' having the next best probability. These types of errors can occur due to faulty HMM model or some human error resulting in change of input sequence.

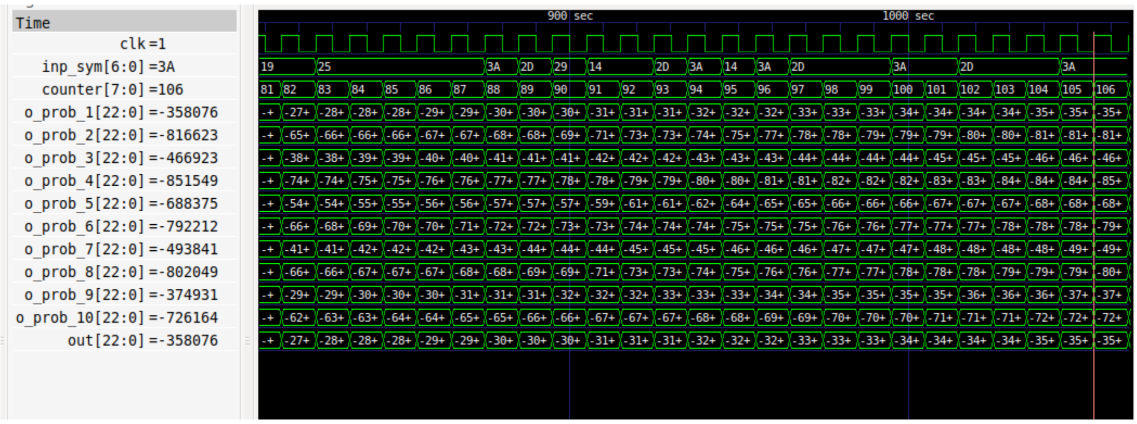


Figure 4.2: Wave forms for a wrongly recognized digit

4.1 Confusion Matrix

The RTL code has been tested against multiple utterances(15) of each word and the results can be presented in the form of a matrix called 'confusion matrix'. Each row in a confusion matrix represent the digit that is actually being recognized and each column represents the number of time this word has been recognized as itself or any of the rest. The following confusion matrix(Table 4.1) was obtained for a total of 150 simulations. From the table it is clear that the digits 'one', 'two', 'three', 'five', 'eight' and 'zero' have shown 100% accuracy where as the digits 'four', 'six', 'seven' and 'nine' have shown 86.7%, 93.3%, 93.3% and 80% efficiencies respectively. Therefore the overall efficiency is around 86%. One way to bring this number up is to test for more number of values along with improving the training models.

4.2 Synthesis

The top module of the RTL designed has been synthesized and implemented in Xilinx tool and the board selected was Zybo. A brief summary on the area oc-

Table 4.1: Table showing the confusion matrix generated by 150 different simulations, fifteen per each word

Digit$\downarrow$	1	2	3	4	5	6	7	8	9	0
1	20	0	0	0	0	0	0	0	0	0
2	0	20	0	0	0	0	0	0	0	0
3	0	0	20	0	0	0	0	0	0	0
4	0	2	0	18	0	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0
6	2	0	0	0	0	18	0	0	0	0
7	1	0	0	0	0	0	19	0	0	0
8	0	0	0	0	0	0	0	20	0	0
9	3	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	20

cupied by the module according to synthesis report is as shown in the following figures 4.3, 4.4 and 4.5. The total number of registers used are seventy 23-bit and one 7-bit. The LUT utilization percentage is high as shown in Fig 4.5 because of the large number of values corresponding to HMM model that needs to be stored.

```
Start RTL Component Statistics
---------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
	   3 Input     23 Bit       Adders := 30
	   2 Input     23 Bit       Adders := 170
	   2 Input     10 Bit       Adders := 1
	   2 Input      9 Bit       Adders := 1
	   2 Input      8 Bit       Adders := 1
	   2 Input      7 Bit       Adders := 1
```

Figure 4.3: Figure shows the number of adders of various bit-lenghts used

```
+---Muxes :
     2 Input    23 Bit      Muxes := 129
     2 Input    17 Bit      Muxes := 20
   449 Input    16 Bit      Muxes := 60
   128 Input    16 Bit      Muxes := 10
     2 Input     1 Bit      Muxes := 130
     7 Input     1 Bit      Muxes := 10
     3 Input     1 Bit      Muxes := 20
     6 Input     1 Bit      Muxes := 10
     5 Input     1 Bit      Muxes := 10
     4 Input     1 Bit      Muxes := 10
```

Figure 4.4: Figure showing the total number of different types of MUX used

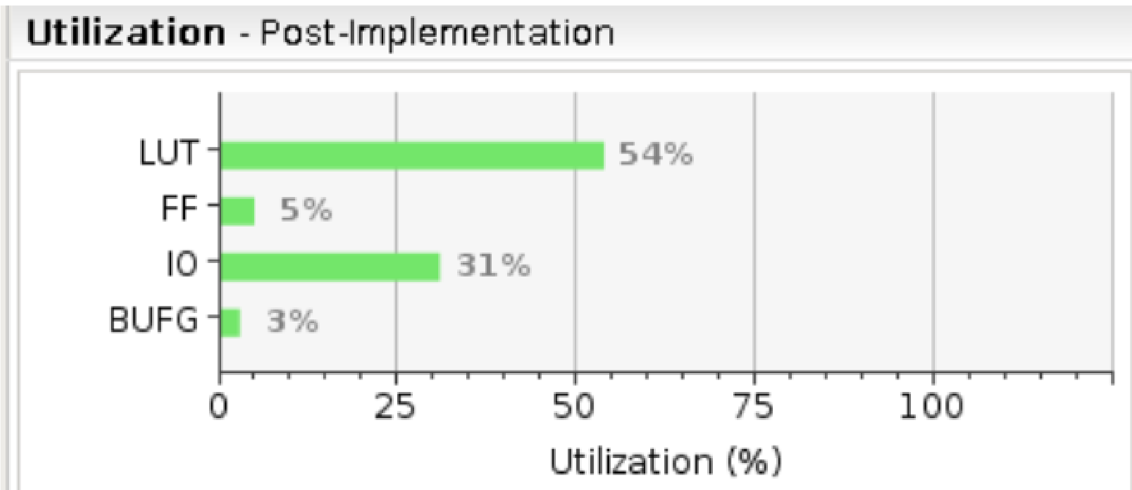


Figure 4.5: Figure showing percentage utilization of hardware on Zybo board

Chapter 5

CONCLUSIONS AND FUTURE WORK

An optimized hardware design of speech decoder for isolated digits has been presented and analyzed.The RTL designed can be easily extended to any number of words in a given set and is also parameterized with respect to the number of bits each register can hold. Dealing with fixed point numbers with appropriate number of bits for each of the storage elements at various stages makes this design faster and lesser area consuming. The obtained results have been comparable to that produced by the MATLAB and hence the goal is achieved.

One of the drawbacks of the model is that we are using parallel hardware for the ten HMM models. This would blow up the LUT utilization as we have seen in Figure 4.5. If we need to design for recognizing twenty words instead of ten then such hardware wouldn't be useful. Instead we can design a model which uses resource sharing but it would take double the number of clock cycles for a twenty word model.

The future work for this project involves extending the idea of replacing floating point with fixed point numbers in dealing the larger and the ultimate goal i.e., sentence recognition system for an Application Specific Integrated Circuit(ASIC) based solution. As a part of this work, a plan to replace floating point operations(by doing operator overloading) in the 'Kaldi' C++ source code for generic sentence recognition has been proposed.

Chapter 6

APPENDIX- REFERENCES

1. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77(2), 257-286*, (1989).
2. https://en.wikipedia.org/wiki/Hidden_Markov_model
3. https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
4. H Bhagawateeswaran, Hardware Accelerator For HMM based speech recognition using approximate computing techniques.(2015)
5. https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf