A B.TECH. PROJECT REPORT ON

IMPLEMENTATION OF SPEECH RECOGNITION IN RESOURCE CONSTRAINED ENVIRONMENTS

UNDER THE GUIDANCE OF

DR. R. MITRA
PROFESSOR

DR. D.K. MEHRA
PROFESSOR

SUBMITTED BY

DEVESH NEMA

KEERTHI RAGHAVAN

SANDEEP AGRAWAL

ELECTRONICS AND COMPUTER ENGINEERING DEPARTMENT

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE – 247667

ACKNOWLEDGEMENTS

We would like to place on record our deep appreciation and gratitude towards our advisors **Prof. R. Mitra** and **Prof. D.K. Mehra** for guiding us in this our project. We also want to express our heartfelt thanks to **Mr. Solomon Raju** who provided us with invaluable help in porting our software solution on to a hardware platform.

We would also like to thank **Prof. Lawrence Rabiner** of the Univ. of California Santa Barbara whose paper inspired us to use Hidden Markov Models in our effort to make an efficient and robust speech recognition system.

Devesh Nema

Keerthi Raghavan

Sandeep Agrawal

CANDIDATES' DECLARATION

We hereby certify that the work which is being presented in this project entitled **'Implementation of Speech Recognition in Resource Constrained Environments'** in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering, IIT Roorkee is an authentic record of our own work carried out over a period of two semesters under the able guidance of Prof. R.Mitra and Prof D.K.Mehra. The matter embodied in this project report has not been submitted by us for the award of any other degree.

May 16th 2006

Devesh Nema **Keerthi Raghavan** **Sandeep Agrawal**

This is to certify that the above statement made by the candidates is correct to the best of our knowledge.

May 16, 2006

(Dr. R.Mitra)
Professor
Dept. of Electronics
& Computer Engg.
IIT Roorkee

(Dr. D.K. Mehra)
Professor
Dept. of Electronics
& Computer Engg.
IIT Roorkee

CONTENTS

ABSTRACT

With the emergence of ubiquitous computing powered by state of the art technology, a constant need has been felt for more convenient methods to input data and commands to a computing device. As the size of the computing devices is decreasing exponentially with time, more sophisticated techniques of human computer interfaces such as speech are fast evolving. The project is an attempt to fulfill the gap between the current speech recognition technology and embedded systems.

The initial objective of the project will be the implementation of a speech recognition engine using Hidden Markov Models. This would involve the design of an efficient MATLAB code on a PC. This phase of the project will involve the development of a limited domain recognition engine spanning numerals only. The subsequent step will involve porting this engine to a Resource Constrained environment such as an FPGA kit. The long term aim would be to eliminate the PC altogether and build a stand-alone system. The recognition engine should contain the capability to be extended to span the entire vocabulary of English language.

Throughout the development, measures will be taken to keep the memory requirement and the processing time of the software as small as possible.

PROJECT PHILOSOPHY

Every Speech Recognition system must be judged on two basic factors which govern its usability – Accuracy and Speed. Unfortunately, one of them almost invariably comes at the cost of the other. A higher accuracy rate implies a wider training sequence and a higher number of iterations in the learning algorithm, all of which would necessarily take a far greater number of clock cycles in a standard processor setting.

The solution, which we have envisaged in the course of this project, is to introduce a degree of parallelism in the methodology - thereby reducing the number of clock cycles required for its implementation. This is possible when we port the recognition phase of the system onto an FPGA. The Viterbi algorithm used for the recognition is inherently dependent on a log-likelihood condition involving only 'add' operations making it ideal for hardware implementation.

On the other hand, accuracy remains an important objective of our project. The precision of the Hidden Markov Model approach that we have used depends almost entirely on the model parameters for every isolated word which needs to be calculated at the very outset. To improve accuracy, we calculate these parameters in a MATLAB environment deriving our results on a large number of test sequences recorded in a typical noisy environment. Once obtained, these parameters are then transferred onto the FPGA, making it capable of functioning independent of the computer. Any new words in the dictionary can be subsequently easily added by calculating the parameters in MATLAB and shifting them to the FPGA.

.

A BRIEF INTRODUCTION TO SPEECH RECOGNITION

Real time continuous speech recognition is a computationally demanding task, and one which tends to benefit from increasing the available computing resources.

A typical speech recognition system starts with a preprocessing stage, which takes a speech waveform as its input, and extracts from it feature vectors or observations which represent the information required to perform recognition. This stage is efficiently performed by software. The second stage is recognition, or decoding, which is performed using a set of phoneme-level statistical models called hidden Markov models (HMMs). Word-level acoustic models are formed by concatenating phone-level models according to a pronunciation dictionary. These word models are then combined with a language model, which constrains the recognizer to recognize only valid word sequences. The decoder stage is computationally expensive.

Although there exist software implementations that are capable of real time performance, there are several reasons why it is worth using hardware acceleration to achieve much faster decoding. Firstly, there exist real telephony-based applications used for call-centers (e.g. the AT&T "How may I help you?" system), where, the speech recognizer is required to process a large number of spoken queries in parallel. Secondly, there are non-real time applications, such as off-line transcription of dictation, where the ability of a single system to process multiple speech streams in parallel may offer a significant financial advantage. Thirdly, the additional processing power offered by an FGPA could be used for real-time implementation of the "next generation" of speech recognition algorithms, which are currently being developed in laboratories. These achieve superior performance but are much more complex and computationally expensive than current methods.

The figure below shows a block diagram of a pattern recognition approach to a continuous speech recognition system.

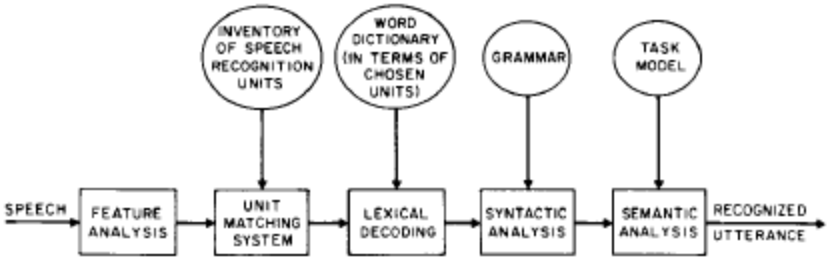


Fig 1: *Block diagram of a continuous speech recognizer*

The key signal processing steps include the following:

1) Feature Analysis: A spectral and/or temporal analysis of the speech signal is performed to give observation vectors which can be used to train the HMMs which characterize various speech sounds.

2) Unit Matching System: First a choice of speech recognition unit must be made. Possibilities include linguistically based sub-word units such as phones (or phone-like units), diphones, demisyllables, and syllables, as well as derivative units such as phenemes, phenones, and acoustic units. Other possibilities include whole word units, and even units which correspond to a group of 2 or more words (e.g., and an, in the, of a, etc). Generally, the less complex the unit (e.g., phones), the fewer of them there are in the language, and the more complicated (variable) their structure in continuous speech. For large vocabulary speech recognition (involving 1000 or more words), the use of sub-word speech units is almost mandatory as it would be quite difficult to record an adequate training set for designing HMMs for units of the size of words or larger. However, for specialized applications (e.g., small vocabulary, constrained task), it is both reasonable and practical to consider the word as a basic speech unit. Independent of the unit chosen for recognition, an inventory of such units must be obtained via training. Typically each

such unit is characterized by some type of HMM whose parameters are estimated from a training set of speech data. The unit matching system provides the likelihoods of a match of all sequences of speech recognition units to the unknown input speech. Techniques for providing such match scores, and in particular determining the best match score (subject to lexical and syntactic constraints of the system) include the stack decoding procedure, various forms of frame synchronous path decoding, and a lexical access scoring procedure.

3) Lexical Decoding: This process places constraints on the unit matching system so that the paths investigated are those corresponding to sequences of speech units which are in a word dictionary (a lexicon). This procedure implies that the speech recognition word vocabulary must be specified in terms of the basic units chosen for recognition. Such a specification can be deterministic (e.g., one or more finite state networks for each word in thevocabu1ary) or statistical (e.g., probabilities attached to the arcs in the finite state representation of words). In the case where the chosen units are words (or word combinations), the lexical decoding step is essentially eliminated and the structure of the recognizer is greatly simplified.

4) Syntactic Analysis: This process, much like lexical decoding, places further constraints on the unit matching system so that the paths investigated are those corresponding to speech units which comprise words (lexical decoding) and for which the words are in a proper sequence as specified by a word grammar. Such a word grammar can again be represented by a deterministic finite state network (in which all word combinations which are accepted by the grammar are enumerated), or by a statistical grammar (e.g., a trigram word model in which probabilities of sequences of 3 words in a specified order are given). For some command and control tasks, only a single word from a finite set of equiprobable is required to be recognized and therefore the grammar is either trivial or unnecessary. Such tasks are often referred to as isolated word speech recognition tasks. For other applications (e.g., digit sequences) very simple grammars are often adequate (e.g., any digit can be spoken and followed by any other digit). Finally there are tasks for which the grammar is a dominant factor and, although it adds a great deal of constraint to the recognition process, it greatly improves recognition performance

by the resulting restrictions on the sequence of speech units which are valid recognition candidates.

5) Semantic Analysis: This process, again like the steps of syntactic analysis and lexical decoding, adds further constraints to the set of recognition search paths. One way in which semantic constraints are utilized is via a dynamic model of the state of the recognizer. Depending on the recognizer state certain syntactically correct input strings are eliminated from consideration. This again serves to make the recognition task easier and leads to higher performance of the system.

1. THEORY OF HIDDEN MARKOV MODELS

The Hidden Markov Model approach is a popular and effective tool to solve the Isolated Word Recognition problem. A brief introduction is provided explaining the basic mathematical theory, followed by an explanation of how the mathematics has a role to play in Speech Recognition.

1.1 DISCRETE MARKOV PROCESSES

Consider a system which may be described at any time as being in one of a set of N distinct states, S_1**,** S_2 . . . , S_N, as illustrated in Fig. **1** (where N = 5 for simplicity). At regularly spaced discrete times, the system undergoes a change of state (possibly back to the same state) according to a set of probabilities associated with the state. We denote the time instants associated with state changes as t = **1, 2,** . . . , and we denote the actual state at time t as q_t.

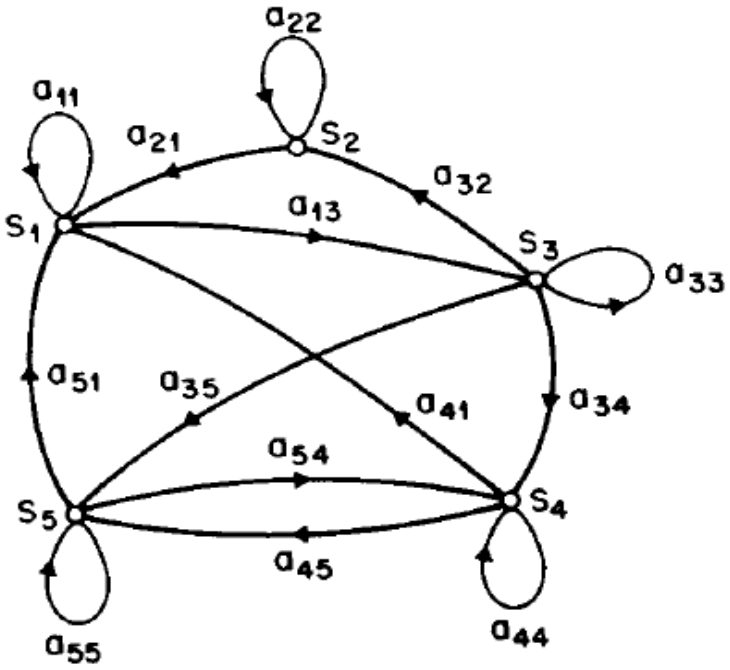


Fig 2 : *A Markov chain with 5 states (labeled S_1 to S_5) with selected state transitions*

A full probabilistic description of the above system would, in general, require specification of the current state (at time *t),* as well as all the predecessor states. For the special case of a discrete, first order, Markov chain, this probabilistic description is truncated to just the current and the predecessor state, i.e.,

$$\mathbf{P[q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots]} \qquad \mathbf{(1)}$$

$$\mathbf{= P[q_t = S_j | q_{t-1} = S_i].}$$

Further more we only consider those processes in which the right-hand side of **(1)** is independent of time, thereby leading to the set of state transition probabilities a_{ij} of the form

$$\mathbf{a_{ij} = P[q_t = S_j | q_{t-1} = S_i].} \qquad \mathbf{1 \leq i, j \leq N} \qquad \mathbf{(2)}$$

with the state transition coefficients having the properties

$$\mathbf{a_{ij} \geq 0} \qquad \mathbf{(3)}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \mathbf{(4)}$$

since they obey standard stochastic constraints.

The above stochastic process could be called an observable Markov model since the output of the process is the set of states at each instant of time, where each state corresponds to a physical (observable) event.

1.2 COIN TOSS MODEL

Assume the following scenario. You are in a room with a barrier (e.g., a curtain) through which you cannot see what is happening. On the other side of the barrier is another person who is performing a coin (or multiple coins) tossing experiment. The other person will not tell you anything about what he is doing exactly; he will only tell you the result of each coin flip. Thus a sequence of hidden coin tossing experiments is performed, with the observation sequence consisting of a series of heads and tails; e.g., a typical observation sequence would be

$$\mathbf{O} = \boldsymbol{O_1}\ \boldsymbol{O_2}\ \mathbf{O}_3 \ldots \boldsymbol{O_T}$$
$$= \text{H, H, T, T, T, H}\ldots$$

where H stands for heads and T stands for tails. Given the above scenario, the problem of interest is how do we build an HMM to explain (model) the observed sequence of heads and tails. The first problem one faces is deciding what the states in the model correspond to, and then deciding how many states should be in the model. One possible choice would be to assume that only a single biased coin was being tossed. In this case we could model the situation with a 2-state model where each state corresponds to a side of the coin (i.e., heads or tails). This model is depicted in Fig. 3(a). In this case the Markov model is observable, and the only issue for complete specification of the model would be to decide on the best value for the bias (i.e., the probability of, say, heads). Interestingly, an equivalent HMM to that of Fig. 3(a) would be a degenerate 1-state model, where the state corresponds to the single biased coin, and the unknown parameter is the bias of the coin.

A second form of HMM for explaining the observed sequence of coin toss outcome is given in Fig. 3(b). In this case there are 2 states in the model and each state corresponds to a different, biased, coin being tossed. Each state is characterized by a probability distribution of heads and tails, and transitions between states are characterized by a state transition matrix. The physical mechanism which accounts for how state transitions are selected could itself be a set of independent coin tosses, or some other probabilistic event.

A third form of HMM for explaining the observed sequence of coin toss outcomes is given in Fig. 3(c). This model corresponds to using 3 biased coins, and choosing from among the three, based on some probabilistic event.

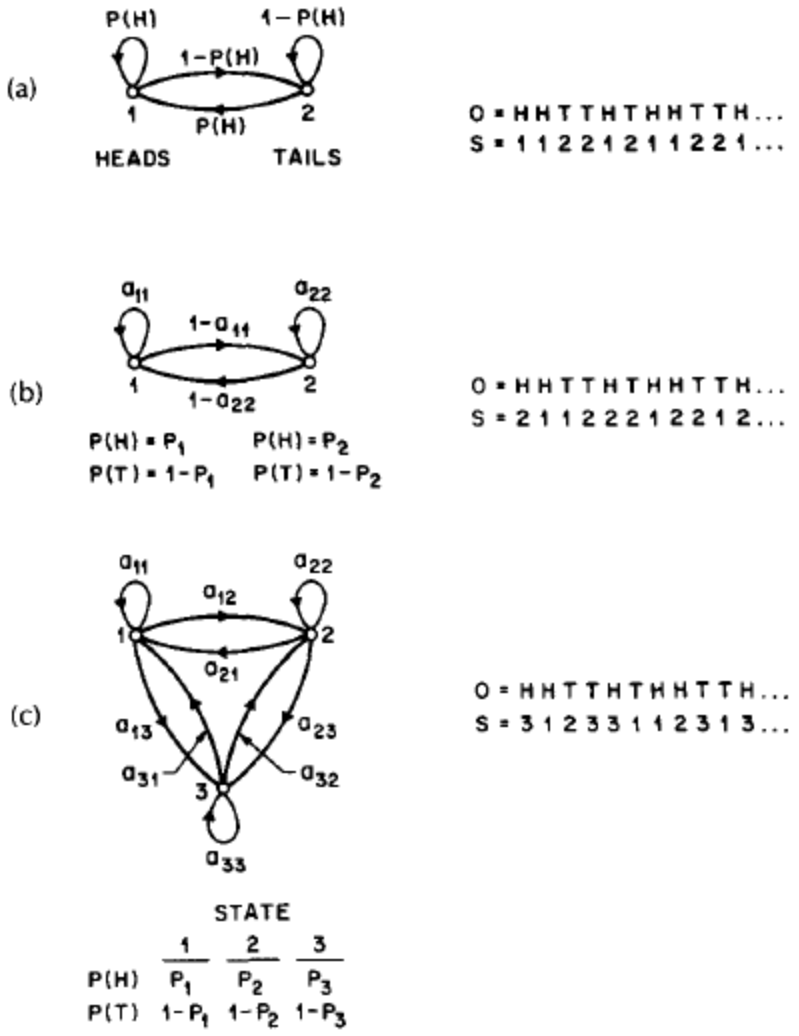


Fig 3: *Three possible Markov models which can account for the results of hidden coin tossing experiments. (a) I-coin model. (b) 2-coins model. (c) 3-coins model.*

Given the choice among the three models shown in Fig. 3 for explaining the observed sequence of heads and tails, a natural question would be which model best matches the actual observations. It should be clear that the simple I-coin model of Fig. 3(a) has only 1 unknown parameter; the 2-coin model of Fig. 3(b) has 4 unknown parameters; and the 3-coin model of Fig. 3(c) has 9 unknown parameters. Thus, with the greater degrees of freedom, the larger HMMs would seem to inherently be more capable of modeling a series of coin tossing experiments than would equivalently smaller models.

1.3 ELEMENTS OF AN HMM

We now formally define the elements of an HMM, and explain how the model generates observation sequences. An HMM is characterized by the following:

1) ***N***, the number of states in the model. Although the states are hidden, for many practical applications there is often some physical significance attached to the states or to sets of states of the model. Generally the states are interconnected in such a way that any state can be re ached from any other state (e.g., an ergodic model). We denote the individual states as $S = \{S_1, S_2, \ldots, S_N\}$, and the state at time ***t*** as q_t.

2) ***M***, the number of distinct observation symbols per state, i.e., the discrete alphabet size. The observation symbols correspond to the physical output of the system being modeled. We denote the individual symbols as

$$V = \{v_1, v_2, \ldots\ldots\ldots\ldots, v_M\}$$

3) The state transition probability distribution $\mathbf{A} = \{a_{ij}\}$ where

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i], \quad 1 \le i, j \le N.$$

For the special case where any state can reach any other state in a single step, we have $a_{ij} > 0$ for all i, j. For other types of HMMs, we would have $a_{ij} = 0$ for one or more (i,j) pairs.

4) The observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where

$$b_j(k) = p[v_k \text{ at } t | q_t = S_j], \qquad 1 \le j \le N, 1 \le k \le M$$

1.4 THE THREE BASIC PROBLEMS FOR HMMS

Given the form of HMM of the previous section, there are three basic problems of interest that must be solved for the model to be useful in real-world applications. These problems are the following:

- **Problem 1**: Given the observation sequence $0 = O_1\ O_2 \ldots O_T$, and a model $\lambda = (A,B,\prod)$, how do we efficiently compute $P(O|\lambda)$, the probability of the observation sequence, given the model?

- **Problem 2:** Given the observation sequence $O = O_1\ O_2 \ldots O_T$, and the model λ, how do we choose a corresponding state sequence $Q = q_1\ q_2 \ldots q_T$ which is optimal in some meaningful sense (i.e., best "explains" the observations)?

- **Problem 3:** How do we adjust the model parameters $\lambda = (A, B, \prod)$ to maximize $P(O|\lambda)$?

Problem 1 is the evaluation problem, namely given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model. Problem 2 attempts to uncover the hidden part of the model, i.e., to find the "correct" state sequence. Problem 3 attempts to optimize the model parameters so as to best describe how a given observation sequence comes about. The observation sequence used to adjust the model parameters is called a training sequence since it is used to "train" the HMM. The training problem is the crucial one for most applications of HMMs, since it allows to optimally adapt model parameters to observed training data-i.e., to create best models for real phenomena.

To fix ideas, consider the following simple ***isolated word speech recognizer***. For each word of a W word vocabulary, we want to design a separate N-state HMM. We represent the speech signal of a given word as a time sequence of coded spectral vectors. We assume that the coding is done using a spectral codebook with M unique spectral vectors; hence each observation is the index of the spectral vector closest (in some spectral sense) to the original speech signal. Thus, for each vocabulary word, we have a training sequence consisting of a number of repetitions of sequences of codebook indices of the word (by one or more talkers). The first task is to build individual word models. This task is done by using the solution to Problem 3 to optimally estimate model parameters for each word model.

To develop an understanding of the physical meaning of the model states, we use the solution to Problem 2 to segment each of the word training sequences into states, and then study the properties of the spectral vectors that lead to the observations occurring in each state. The goal here would be to make refinements on the model (e.g., more states, different codebook size, etc.) so as to improve its capability of modeling the spoken word sequences. Finally, once the set of $\boldsymbol{W}$ HMMs has been designed and optimized and thoroughly studied, recognition of an unknown word is performed using the solution to Problem 1 to score each word model based upon the given test observation sequence, and select the word whose model score is highest (i.e. the highest likelihood).

1.5 ISOLATED WORD RECOGNITION

Assume we have a vocabulary of V words to be recognized and that each word is to be modeled by a distinct HMM. Further assume that for each word in the vocabulary we have a training set of *K* occurrences of each spoken word (spoken by 1 or more talkers) where each occurrence of the word constitutes an observation sequence, where the observations are some appropriate representation of the (spectral and/or temporal) characteristics of the word. In order to do isolated word speech recognition, we must perform the following:

1) For each word v in the vocabulary, we must build HMM λ^v, i.e., we must estimate the model parameters ***(A, B,*** $\prod$) that optimize the likelihood of the set observation vectors for the v^{th} word.

2) For each unknown word which is to be recognized, the processing shown below must be carried out, namely measurement of the observation sequence $0 = \{O_1, O_2 \ldots\ldots O_T\}$, via a feature analysis of the speech corresponding to the word; followed by calculation of model likelihoods for all possible models, $P(O|\lambda^v)$, $1 \leq v \leq V$; followed by selection of the word whose model likelihood is highest, i.e.,

$$v^* = \underset{1 \leq v \leq V}{\operatorname{argmax}} [P(O \mid \lambda^v)].$$

The probability computation step is generally performed using the Viterbi algorithm (i.e., the maximum likelihood path is used and requires on the order of $V*N^2*T$ computations. For modest vocabulary sizes, e.g., V = 100 words, with an $N = 5$ state model, and $T = 40$ observations for the unknown word, a total of 10^5 computations is required for recognition (where each computation is a multiply, and add, and a calculation of observation density, *b(O).* Clearly this amount of computation is modest as compared to the capabilities of most modern signal processor chips.

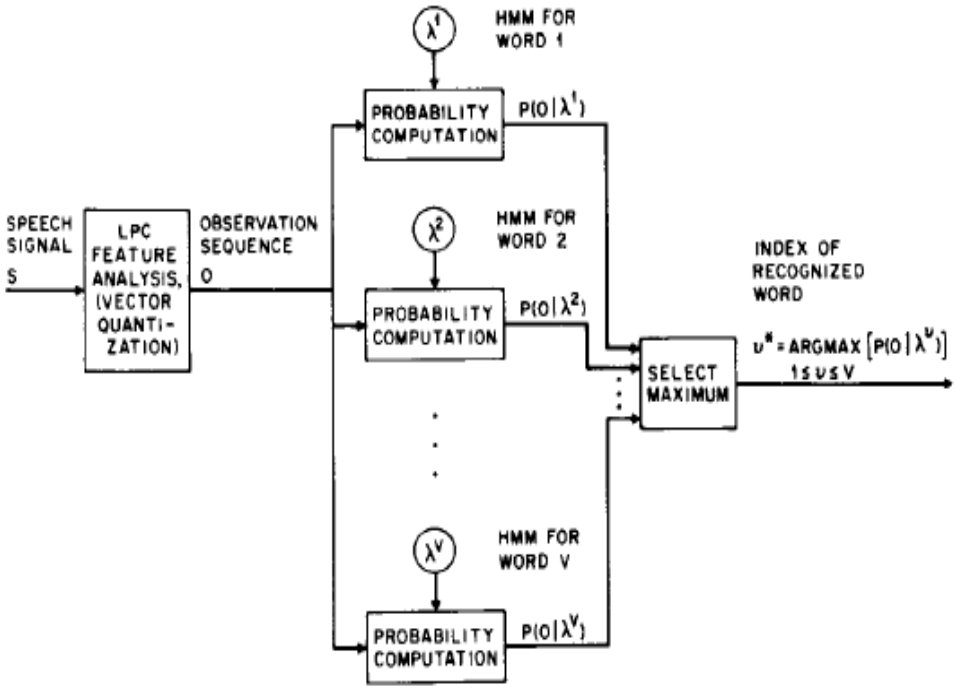


Fig. 4: *Block diagram of an isolated word HMM recognizer*

The overall system is a block processing model in which a frame of N_A samples is processed and a vector of features O_t, is computed. The steps in the processing are as follows:

1) **Pre-emphasis:** The digitized speech signal is processed by a first-order digital network in order to spectrally flatten the signal.

2) **Blocking into Frames:** Sections of N_A consecutive speech samples are used as a single frame. Consecutive frames are spaced M_A samples apart.

3) **Frame Windowing:** Each frame is multiplied by an N_A sample window w(n) so as to minimize the adverse effects of chopping an N_A sample section out of the running speech signal.

4) **Autocorrelation Analysis:** Each windowed set of speech samples is auto-correlated to give a set of (p + 1) coefficients, where p is the order of the desired LPC analysis .

5) **LPC Analysis:** For each frame, a vector of LPC coefficients is computed from the autocorrelation vector using a Levinson or a Durbin recursion method. An LPC derived cepstral vector is then computed up to the Qth component, where Q > p .

2. IMPLEMENTATION OF THE ISOLATED WORD RECOGNIZER

SYSTEM SCHEMATIC

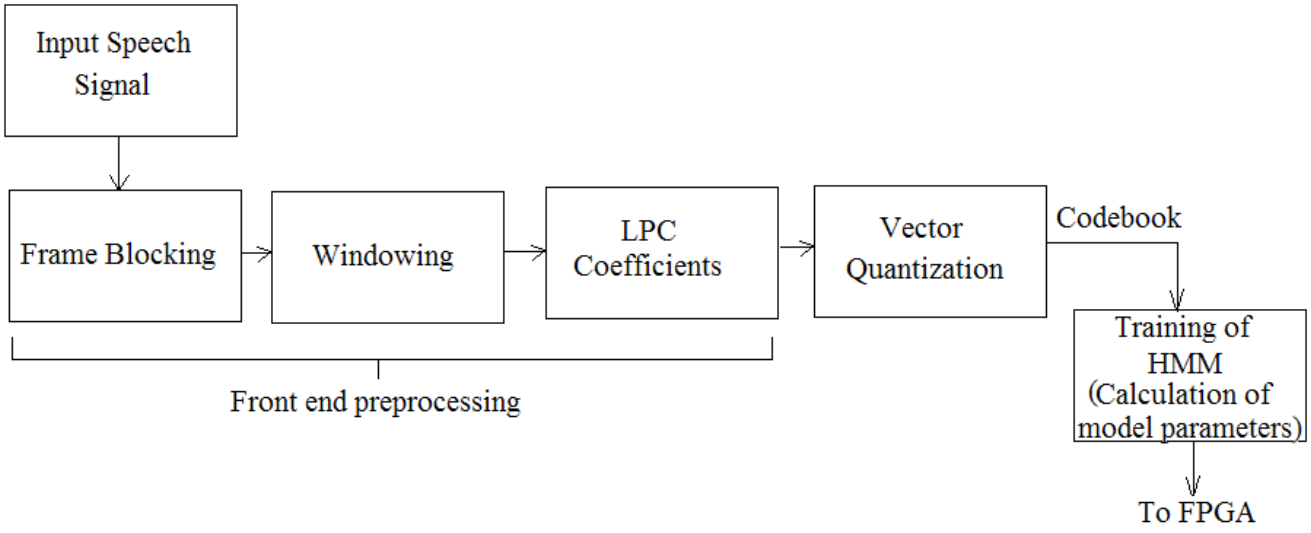


Fig. 5: *Training Phase: Preprocessing*

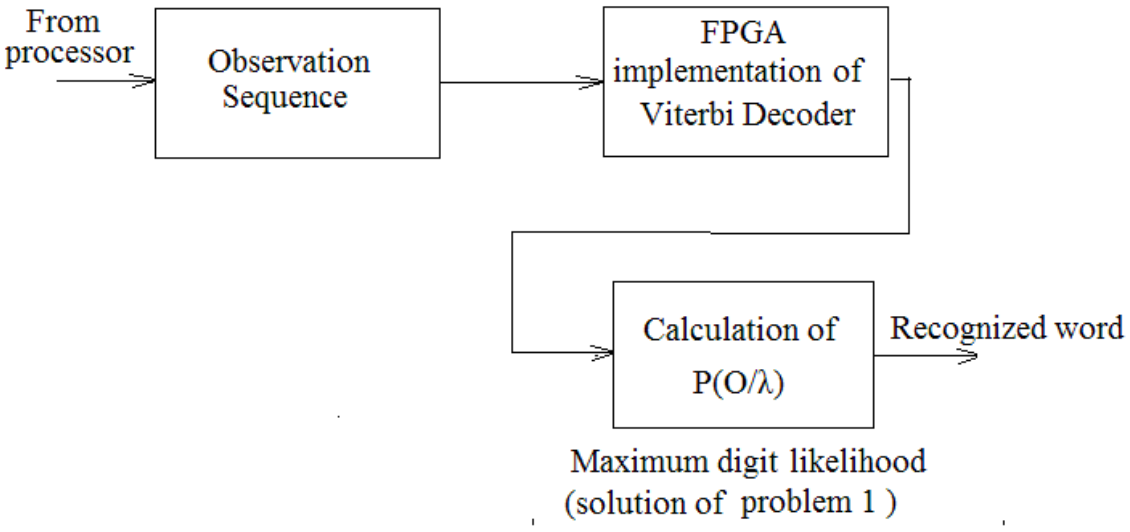


Fig. 6: *Testing Phase*

As is clear from the System Schematic shown in Fig. 4 and Fig. 5, processing of the preliminary speech samples has an important role to play to solve Problem 3 of the Hidden Markov Models and to generate appropriate model parameters.

The first task is to obtain the Linear Predictive Coding coefficients of the speech samples.

2.1 LINEAR PREDICTIVE CODING (LPC)

Linear predictive coding (LPC) is used in audio signal processing and speech processing for representing the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model. It is a way of encoding the information in a speech signal into a smaller space for transmission over a restricted channel. It is one of the most powerful speech analysis techniques, and one of the most useful methods for encoding good quality speech at a low bit rate and provides extremely accurate estimates of speech parameters. LPC has applications in filter design and speech coding.

LPC encodes a signal by finding a set of weights on earlier signal values that can predict the next signal value. It determines the coefficients of a forward linear predictor by minimizing the prediction error in the least squares sense. For a p^{th}- order linear predictor, the current value of the real-valued time series X(n) is based on past samples.

$$Xp(n) = -A(2)*X(n-1) - A(3)*X(n-2) - ... - A(N+1)*X(n-N)$$

such that the sum of the squares of the errors

$$err(n) = X(n) - Xp(n)$$

is minimized. And the LPC coefficients are given by

$$A(1), A(2), ... A(N+1)$$

Training Procedure:

Isolated words such as ‘zero’ and ‘one’ were spoken by five different speakers, each uttering a word 20 times. These words were recorded in the 16 bit-mono PCM format at a sampling rate of 10,000 samples per second. The acquired speech files were processed using the Goldwave software for Noise-Reduction. One of the edited speech waveforms for each shown in Figure 6 and 7.

On the edited speech samples, LPC was carried out using the Durbin’s method to extract LPC coefficients frame by frame. The speech samples are framed with a frame size of N = 420 samples (=42 milliseconds). Consecutive frames are spaced M = 180 samples apart (= 8 milliseconds), corresponding to a frame overlap of 240 samples (=24 milliseconds). Then each frame is multiplied by a N sample Hamming Window W(n), where

$$W(n)= 0.54 – 0.46 \cos (2\prod n/(N-1))$$

Hamming window is very useful in speech like waveforms to smoothen the ends of the frame. Each windowed set of speech samples is auto-correlated to give a set of ‘p+1’ coefficients, where ‘p’ is the order of the desired LPC vector. We have chosen p=8. One such LPC coefficients for each is shown in Figure 8 and 9.

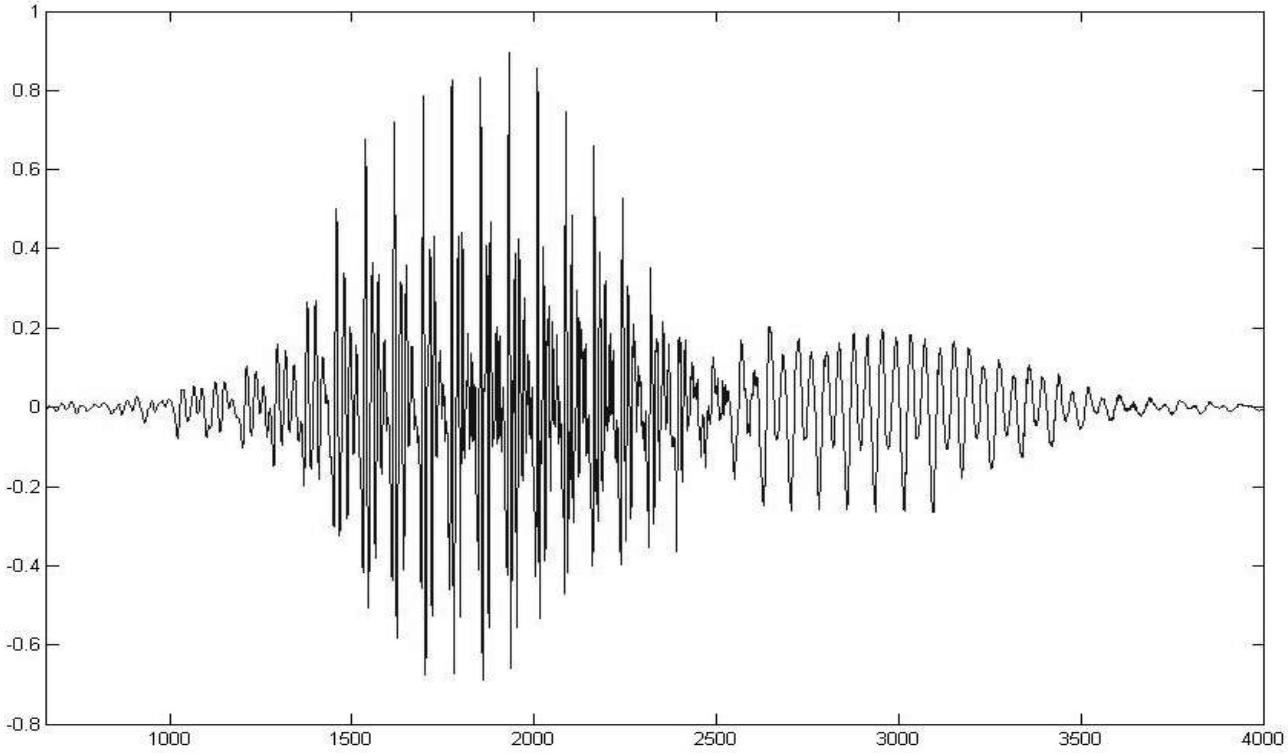


Fig 7 *: Edited Speech waveform for the word 'one'*

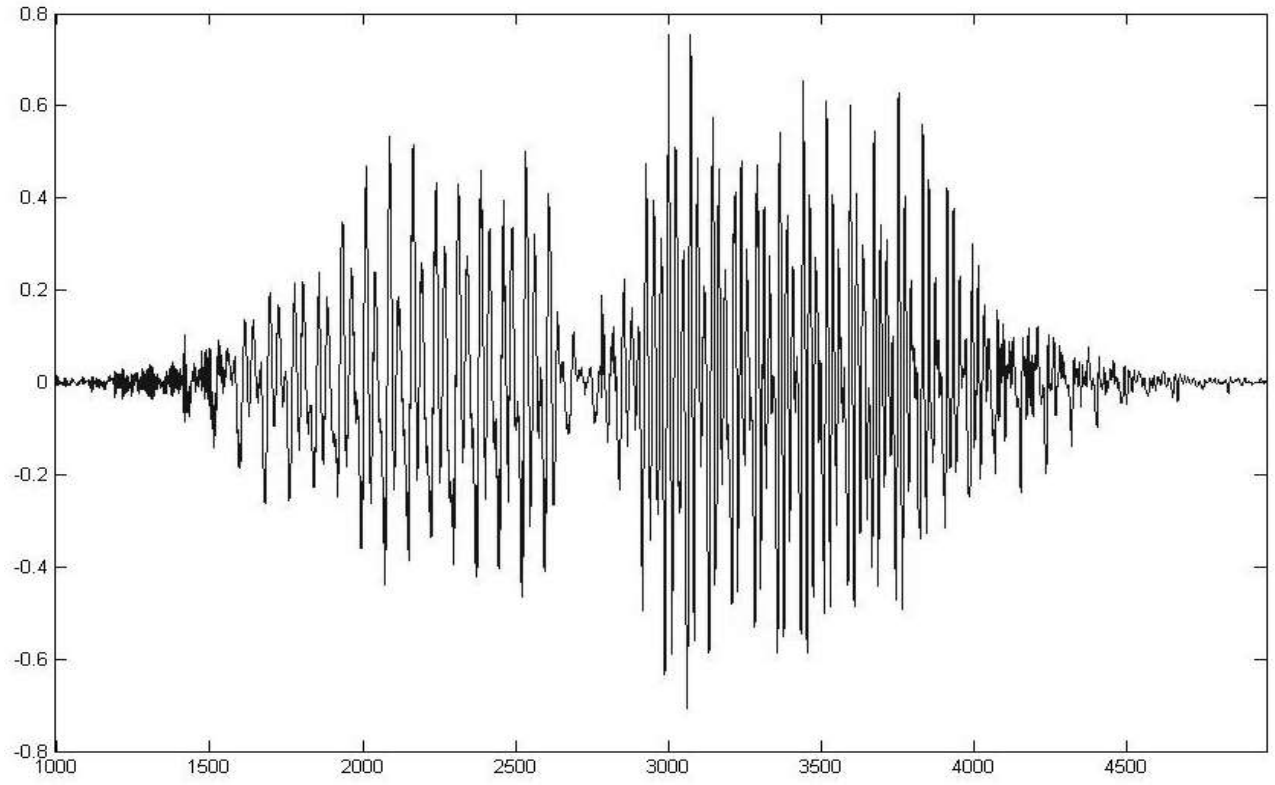


Fig 8 *: Edited Speech waveform for the word 'zero'*

```
##############################################################################

    1         2         3         4         5         6         7         8         9

 1.0000 -1.3077 -0.5619  0.8847  0.4551 -0.2968 -0.0585 -0.2578  0.1999
 1.0000 -1.7458 -0.1433  1.3635  0.0045 -0.5061  0.1535 -0.3348  0.2411
 1.0000 -2.0050  0.6155  0.7700 -0.1822 -0.0676  0.0345 -0.4385  0.3089
 1.0000 -2.3305  1.4073  0.2596 -0.1543 -0.1859 -0.0502  0.0111  0.0656
 1.0000 -2.1658  1.0816  0.3442 -0.0552 -0.1126 -0.0951 -0.1250  0.1515
 1.0000 -2.1666  1.0504  0.3354  0.0031 -0.0950 -0.1442 -0.0802  0.1162
 1.0000 -2.9737  3.0245 -1.0388  0.2476 -0.3578 -0.2492  0.5698 -0.2098
 1.0000 -2.9445  3.1708 -1.3172  0.1180  0.1937 -0.6278  0.6261 -0.1838
 1.0000 -2.3394  1.8618  0.0477 -1.0579  0.9199 -0.2382 -0.2865  0.2488
 1.0000 -2.2222  1.6832  0.3513 -1.4306  1.0697 -0.1239 -0.3766  0.2781
 1.0000 -2.0828  1.5547  0.4659 -1.7202  1.2818  0.1855 -0.8981  0.5323
 1.0000 -2.1183  1.7245  0.1750 -1.5975  1.4240 -0.1185 -0.6402  0.4288
 1.0000 -2.2568  2.0423 -0.3210 -1.1933  1.3696 -0.6241  0.0617  0.0757
 1.0000 -2.4026  2.3403 -0.9657 -0.4410  0.9887 -0.7721  0.2885  0.0049
 1.0000 -2.3273  2.3639 -1.3191 -0.3177  1.3854 -1.1661  0.4082  0.0166
 1.0000 -2.2936  2.3761 -1.5908  0.1291  1.0316 -1.0222  0.4473 -0.0346
 1.0000 -2.3557  2.6114 -1.9884  0.5955  0.6046 -0.7528  0.3178  0.0096
 1.0000 -2.0860  2.3076 -2.2278  1.3670 -0.5831  0.4066 -0.3252  0.1984
 1.0000 -1.7371  1.5700 -1.4955  0.9819 -0.8223  1.0239 -0.8915  0.4479
 1.0000 -1.4818  1.1604 -1.1580  0.7947 -0.9220  1.1990 -1.1162  0.6095
 1.0000 -1.4709  1.0370 -1.0235  0.6659 -0.8136  1.1919 -1.0047  0.4815
 1.0000 -1.3395  0.7988 -0.9425  0.5932 -0.7214  1.1543 -0.8889  0.3908
 1.0000 -1.2992  0.7802 -0.9088  0.5142 -0.7245  1.1582 -0.9370  0.4568
 1.0000 -1.2915  0.5883 -0.8452  0.6643 -0.6034  1.0644 -0.8862  0.3392
 1.0000 -1.2859  0.3481 -0.6807  0.7858 -0.4639  1.0477 -0.8925  0.1614
 1.0000 -1.1864  0.1118 -0.7142  1.0367 -0.2953  0.8538 -0.9216  0.1334
 1.0000 -1.4530  0.2983 -0.6737  1.3870 -0.5045  0.6788 -1.1506  0.4374
 1.0000 -1.5011  0.0748 -0.3940  1.6020 -0.4564  0.1421 -1.0373  0.5888
 1.0000 -1.5321 -0.0026 -0.2257  1.5929 -0.3399 -0.2658 -0.7715  0.5631
 1.0000 -1.5679  0.0144 -0.0940  1.4245 -0.2338 -0.4741 -0.5227  0.4714
 1.0000 -1.5739 -0.0580  0.1012  1.1887  0.1010 -0.8566 -0.3328  0.4481
 1.0000 -1.4864 -0.2506  0.1667  1.2283  0.1728 -0.8175 -0.5606  0.5659

##############################################################################
```

Fig 9 *: LPC coefficients for the word 'one'*

```
##########################################################################################

    1         2         3         4         5         6         7         8         9


1.0000   0.5313 -0.8877 -0.5673 -0.2694 -0.2842  0.2679  0.4049  0.1903
1.0000   0.5745 -0.9665 -0.7177 -0.2092 -0.2113  0.2186  0.4487  0.2165
1.0000   0.7818 -0.8566 -1.0625 -0.3162 -0.0389  0.1899  0.4542  0.3263
1.0000   0.6829 -1.3030 -1.1772  0.2264  0.4559  0.1656  0.1281  0.1226
1.0000   0.7241 -1.3127 -1.5542  0.2379  1.0650  0.2554 -0.1672  0.0648
1.0000   0.8927 -0.9011 -1.3186 -0.3449  0.2267  0.2243  0.3364  0.2922
1.0000   0.9316 -0.6826 -1.0524 -0.5334 -0.3093  0.0086  0.6411  0.5205
1.0000   0.5362 -0.6908 -0.7597 -0.4103 -0.2708 -0.0191  0.5386  0.4249
1.0000  -0.7088 -0.3423 -0.2202 -0.0104  0.0347  0.2155  0.4972 -0.3469
1.0000  -1.2746  0.7445 -0.9158  0.5876 -0.5303  0.3591  0.3268 -0.1811
1.0000  -1.5611  1.4318 -1.7412  1.4713 -1.3641  1.0708 -0.2352  0.0357
1.0000  -1.7018  1.8128 -2.2099  1.9447 -1.7949  1.4188 -0.5441  0.1782
1.0000  -1.8228  2.1448 -2.4827  2.2576 -2.2308  1.7462 -0.8593  0.3736
1.0000  -2.0473  2.5211 -2.8031  2.6405 -2.5672  1.9924 -0.9426  0.3363
1.0000  -1.7461  1.9235 -2.1477  1.6937 -1.3137  0.8376 -0.2643  0.1562
1.0000  -2.3479  3.1659 -3.5559  3.5271 -3.1856  2.3033 -1.1469  0.4018
1.0000  -2.6660  3.6600 -3.9104  3.9869 -3.7897  2.8283 -1.3968  0.3998
1.0000  -2.5085  2.9389 -2.7459  2.9736 -2.8341  1.7858 -0.6951  0.1884
1.0000  -2.7896  3.8857 -4.2380  4.4663 -4.1218  3.0732 -1.6893  0.5220
1.0000  -2.8743  4.1470 -4.6995  5.0079 -4.5244  3.3106 -1.8478  0.5813
1.0000  -2.8354  3.9467 -4.3604  4.6728 -4.2014  2.9932 -1.6420  0.5183
1.0000  -2.9274  4.2683 -4.8670  5.1990 -4.8178  3.7624 -2.2288  0.6963
1.0000  -2.8519  4.0968 -4.6188  4.8333 -4.5205  3.7246 -2.3584  0.7586
1.0000  -2.6505  3.5656 -3.9055  4.0428 -3.8099  3.3187 -2.2244  0.7293
1.0000  -2.2129  2.4440 -2.4184  2.4436 -2.2803  1.9750 -1.1630  0.2743
1.0000  -2.5055  3.1331 -3.2201  3.1615 -2.8453  2.4533 -1.5494  0.4472
1.0000  -2.7328  3.7613 -4.0449  4.0346 -3.6970  3.1563 -2.0545  0.6580
1.0000  -2.7702  3.8100 -4.0563  4.0048 -3.6683  3.0042 -1.8234  0.5389
1.0000  -2.6745  3.5100 -3.5540  3.2662 -2.8748  2.4368 -1.5208  0.4198
1.0000  -2.4429  2.6112 -1.9442  1.3917 -1.1153  0.9420 -0.5427  0.1027
1.0000  -2.0482  1.4797 -0.4723 -0.0014 -0.0088  0.2708 -0.1909 -0.0256
1.0000  -1.6257  0.6035  0.1362 -0.1512  0.0214  0.1276  0.0634 -0.1660

##########################################################################
```

Fig 10 *: LPC coefficients for the word 'zero'*

Once the LPC coefficients are obtained, we quantize them using a scheme known as Vector Quantization.

2.2 VECTOR QUANTIZATION (VQ)

Vector quantization (VQ) is a lossy data compression method based on the principle of block coding. It is a fixed-to-fixed length algorithm. It is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region is called a *cluster* and can be represented by its center called a *codeword*. The collection of all codewords is called a *codebook.*

Design Problem:

The VQ design problem can be stated as follows. Given a vector source with its statistical properties known, given a distortion measure, and given the number of code-vectors, find a codebook and a partition which result in the smallest average distortion.

Assuming that there is a *training sequence* consisting of ***M*** source vectors:

$$\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}.$$

M is assumed to be sufficiently large so that all the statistical properties of the source are captured by the training sequence. The source vectors are ***k***-dimensional, e.g.,

$$\mathbf{x}_m = (x_{m,1}, x_{m,2}, \ldots, x_{m,k}), \quad m = 1, 2, \ldots, M.$$

Let ***N*** be the number of code-vectors and let

$$\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N\},$$

represent the codebook. Each code-vector is ***k***-dimensional, e.g.,

$$\mathbf{c}_n = (c_{n,1}, c_{n,2}, \ldots, c_{n,k}), \quad n = 1, 2, \ldots, N.$$

Let $\boldsymbol{S_n}$ be the encoding region associated with code-vector $\boldsymbol{c_n}$ and let

$$\mathcal{P} = \{S_1, S_2, \ldots, S_N\},$$

denote the partition of the space. If the source vector $\mathbf{x}_m$ is in the encoding region $\boldsymbol{S_n}$, then its approximation (denoted by $\boldsymbol{Q(x_m)}$) is $\boldsymbol{c_n}$:

$$Q(\mathbf{x}_m) = \mathbf{c}_n, \quad \text{if } \mathbf{x}_m \in S_n.$$

Assuming a squared-error distortion measure, the average distortion is given by:

$$D_{ave} = \frac{1}{Mk} \sum_{m=1}^{M} ||\mathbf{x}_m - Q(\mathbf{x}_m)||^2,$$

Where

$$||\mathbf{e}||^2 = e_1^2 + e_2^2 + \ldots + e_k^2$$

The design problem can be succinctly stated as follows: Given ***T*** and ***N***, find ***C*** and ***P*** such that $\mathbf{D_{ave}}$ is minimized.

Optimality Criteria

If ***C*** and ***P*** are a solution to the above minimization problem, then it must satisfied the following two criteria.

- **Nearest Neighbor Condition:**

$$S_n = \{\mathbf{x} : ||\mathbf{x} - \mathbf{c}_n||^2 \leq ||\mathbf{x} - \mathbf{c}_{n'}||^2 \quad \forall n' = 1, 2, \ldots, N\}$$

  This condition says that the encoding region $\boldsymbol{S_n}$ should consists of all vectors that are closer to $\boldsymbol{c_n}$ than any of the other code-vectors.

- **Centroid Condition:**

$$\mathbf{c}_n = \frac{\sum_{\mathbf{x}_m \in S_n} \mathbf{x}_m}{\sum_{\mathbf{x}_m \in S_n} 1} \quad n = 1, 2, \ldots, N$$

  This condition says that the code-vector $\boldsymbol{c_n}$ should be average of all those training vectors that are in encoding region $\boldsymbol{S_n}$. In implementation, one should ensure that at least one training vector belongs to each encoding region (so that the denominator in the above equation is never 0).

LBG Design Algorithm

Given ***T*** . Let ε>0 be a ``small'' number.

1. Let ***N=1*** and

$$\mathbf{c}_1^* = \frac{1}{M}\sum_{m=1}^{M}\mathbf{x}_m.$$

   Calculate

$$D_{ave}^* = \frac{1}{Mk}\sum_{m=1}^{M}||\mathbf{x}_m - \mathbf{c}_1^*||^2.$$

2. **Splitting:** For i=1,2 …. , N set

$$\begin{aligned}\mathbf{c}_i^{(0)} &= (1+\epsilon)\mathbf{c}_i^*,\\ \mathbf{c}_{N+i}^{(0)} &= (1-\epsilon)\mathbf{c}_i^*.\end{aligned}$$

   Set ***N=2N***.

3. **Iteration:** Let

$$D_{ave}^{(0)} = D_{ave}^*.$$

4. Set the iteration index i=0.
   i. For m=1,2,…..,M, find the minimum value of

$$||\mathbf{x}_m - \mathbf{c}_n^{(i)}||^2,$$

   over all n=1,2 …. , N. Let n* be the index which achieves the minimum. Set

$$Q(\mathbf{x}_m) = \mathbf{c}_{n^*}^{(i)}.$$

   ii. For n=1,2 …. , N, update the code-vector

$$\mathbf{c}_n^{(i+1)} = \frac{\sum_{Q(\mathbf{x}_m)=\mathbf{c}_n^{(i)}}\mathbf{x}_m}{\sum_{Q(\mathbf{x}_m)=\mathbf{c}_n^{(i)}}1}$$

iii. Set i = i+1.

iv. Calculate

$$D_{ave}^{(i)} = \frac{1}{Mk} \sum_{m=1}^{M} ||\mathbf{x}_m - Q(\mathbf{x}_m)||^2.$$

v. If

$$(D_{ave}^{(i-1)} - D_{ave}^{(i)})/D_{ave}^{(i-1)} > \epsilon$$

,

go back to Step (i).

Set

$$D_{ave}^{*} = D_{ave}^{(i)}$$

For n=1,2,…..,N , set

$$\mathbf{c}_n^{*} = \mathbf{c}_n^{(i)}$$

as the final code-vectors.

5. Repeat Steps 3 and 4 until the desired number of code-vectors is obtained.

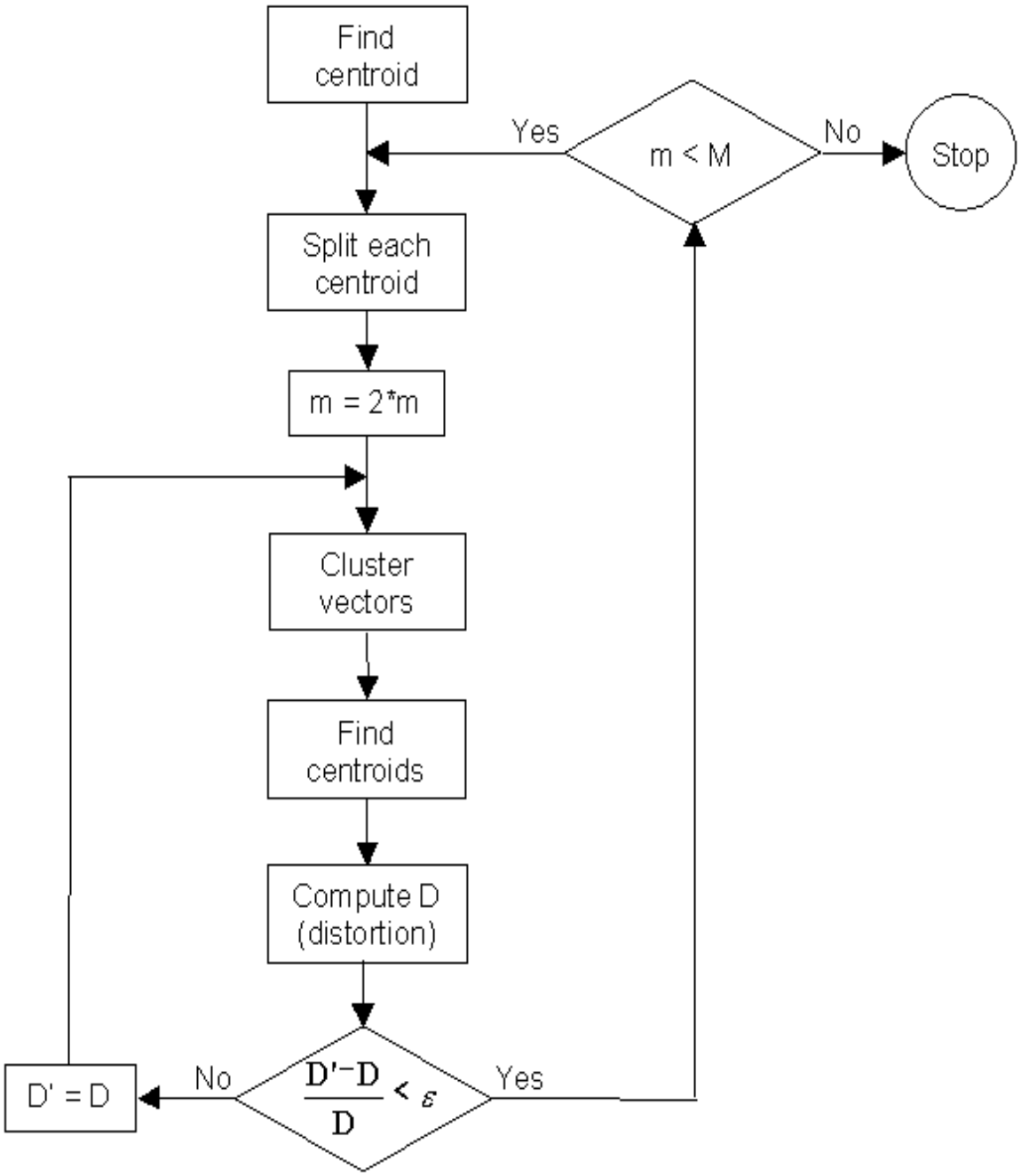


***Fig. 11** : LBG Algorithm flowchart*

Procedure

From the database of each word consisting of 100 files each, observation vectors were extracted. Then these observation vectors are clustered using the LBG Algorithm, to get symbols called codebook of size M. All codebooks are of size 32 symbols with dimension 9. The codebooks for 'zero' and 'one' are shown in Figure 11 and 12.

```
#################################################################################

    1         2         3         4         5         6         7         8         9


1.0000 -1.2575  0.8432 -1.0486  0.7701 -0.3930  0.2832 -0.2388  0.2167
1.0000 -1.5756  0.7984 -0.5703  0.3970 -0.0856  0.4063 -0.6330  0.3235
1.0000 -1.7207  1.2317 -0.8803  0.7329 -0.7787  0.9796 -0.9194  0.4415
1.0000 -1.5206  0.7419 -0.3599  0.3576 -0.6088  0.8968 -0.7809  0.3504
1.0000 -0.3427 -0.0745 -0.3995 -0.1213 -0.0797  0.2038 -0.0434  0.2111
1.0000 -1.1583 -0.0724  0.0809  0.3371 -0.0447 -0.1499 -0.0082  0.1005
1.0000 -1.6127  0.3151  0.6702 -0.5321  0.2133  0.3895 -0.7347  0.3619
1.0000 -1.6454  0.5172  0.0900  0.0165  0.1222  0.0374 -0.1958  0.1016
1.0000 -2.0131  1.4187 -0.4377  0.5515 -0.7588  0.4123 -0.0814  0.0180
1.0000 -2.3565  1.4738  0.2524  0.4048 -1.5285  0.7179  0.2774 -0.2058
1.0000 -2.0465  1.0967  0.0633  0.3093 -0.3718 -0.3388  0.4726 -0.1247
1.0000 -2.1447  1.1430  0.4028 -0.2025 -0.3082  0.0054  0.2038 -0.0629
1.0000 -1.9399  1.4793 -0.8397  0.3192  0.0170  0.1368 -0.3185  0.1987
1.0000 -1.8504  1.3967 -0.8650  0.2581  0.3919 -0.6605  0.4683 -0.0809
1.0000 -2.4519  2.2113 -0.5288 -0.7250  0.9224 -0.4409 -0.0373  0.1066
1.0000 -2.1719  1.5070 -0.2063 -0.3687  0.3960  0.0837 -0.4285  0.2208
1.0000 -2.1912  2.2371 -1.9959  2.1609 -1.9236  1.2886 -0.7344  0.2679
1.0000 -2.5948  3.0097 -2.4294  1.8545 -1.2570  0.5370 -0.1232  0.0451
1.0000 -1.9984  1.7595 -1.2968  1.2679 -1.2384  0.9694 -0.6459  0.2721
1.0000 -2.2273  2.1497 -1.5188  1.2119 -0.9160  0.4396 -0.1499  0.0715
1.0000 -2.5492  3.1033 -2.9803  2.9505 -2.5814  1.7688 -0.9757  0.3395
1.0000 -3.0330  4.2763 -3.9452  2.7007 -1.3535  0.4763 -0.1485  0.0598
1.0000 -3.0304  4.5118 -4.8945  4.7006 -3.9742  2.7111 -1.3292  0.3548
1.0000 -3.3493  5.3135 -5.7144  4.6940 -2.8729  1.2840 -0.4307  0.0914
1.0000 -2.1526  2.0653 -1.4189  0.7086 -0.1895  0.0105 -0.0968  0.1353
1.0000 -2.4786  2.7642 -2.0572  1.0813 -0.2287 -0.2383  0.2226 -0.0221
1.0000 -2.1513  2.1239 -1.5918  0.6076  0.4107 -0.9799  0.9146 -0.2926
1.0000 -2.5246  2.5903 -1.3889  0.0930  0.6397 -0.5636  0.1317  0.0531
1.0000 -2.8213  3.4130 -1.9112 -0.3514  1.7979 -1.8206  0.9525 -0.2040
1.0000 -2.7578  3.2296 -2.0420  0.3309  0.8641 -1.0251  0.5102 -0.0803
1.0000 -2.9145  3.8804 -3.1324  1.3514  0.2550 -0.8278  0.5191 -0.0984
1.0000 -3.1858  4.4229 -3.3439  0.8777  1.2720 -1.9022  1.1635 -0.2816

#################################################################################
```

***Fig. 12** : Codebook for the word 'one'*

```
##########################################################################################

    1        2        3        4        5        6        7        8        9

1.0000 -1.9519  2.0155 -2.0844  2.0131 -1.7087  1.3534 -0.9211  0.4140
1.0000 -2.1958  1.9860 -1.6532  2.0523 -2.0408  1.5459 -1.0440  0.4093
1.0000 -2.2331  2.4288 -2.1596  1.8040 -1.4175  0.9082 -0.4374  0.1828
1.0000 -2.6094  2.9254 -2.1070  1.4916 -1.0958  0.4977 -0.0697  0.0212
1.0000 -2.3114  2.7360 -2.7469  2.6497 -2.4032  1.8050 -0.9791  0.3361
1.0000 -2.6481  3.4945 -3.4659  3.1042 -2.5680  1.7832 -0.9069  0.2877
1.0000 -2.6651  3.4702 -3.2312  2.5244 -1.7801  1.0106 -0.3904  0.1266
1.0000 -2.6847  3.4352 -2.9289  1.7654 -0.6953  0.0746  0.1064 -0.0095
1.0000 -2.9516  4.3267 -4.8506  5.0373 -4.6986  3.7571 -2.2368  0.6895
1.0000 -3.0115  4.4719 -4.9246  4.9060 -4.3660  3.2538 -1.7825  0.5255
1.0000 -3.0499  4.6505 -5.3291  5.5176 -5.1267  4.0429 -2.3302  0.6922
1.0000 -3.2194  5.2363 -6.1911  6.3667 -5.8487  4.5302 -2.5372  0.7414
1.0000 -2.7635  3.7952 -4.0862  4.1879 -3.8985  3.1363 -1.8727  0.5815
1.0000 -2.9161  4.2620 -4.6954  4.6051 -3.9336  2.7563 -1.4259  0.4122
1.0000 -2.6720  3.4691 -3.5663  3.5996 -3.2696  2.4688 -1.3962  0.4364
1.0000 -2.8708  4.0514 -4.1815  3.8008 -3.0408  1.9437 -0.8878  0.2414
1.0000  0.8883 -0.5233 -1.4750 -0.7711  0.0592  0.4552  0.3759  0.2042
1.0000  0.6523 -1.4146 -1.6025  0.2779  1.0241  0.3974 -0.1483 -0.0796
1.0000  0.0238 -0.7745 -0.4193 -0.0190 -0.1872  0.0900  0.2865  0.1587
1.0000 -0.1657 -1.4687 -0.3129  0.8513  0.2748 -0.0185 -0.0572 -0.0392
1.0000 -0.8845 -0.9400  0.8559  0.1623 -0.5982  0.4186  0.2821 -0.2512
1.0000 -1.8239  0.1043  1.4419  0.0289 -1.2172  0.1638  0.5814 -0.2243
1.0000 -0.8995 -0.4178  0.1262  0.0119 -0.0046  0.2725  0.0826 -0.1190
1.0000 -1.4672  0.2459  0.1521  0.0340 -0.0586  0.3102 -0.2284  0.0474
1.0000 -1.1334  0.8803 -1.1370  0.6860 -1.0758  1.0226 -0.3819  0.2927
1.0000 -1.7026  0.8123 -0.4460  0.7648 -0.7541  0.7217 -0.5393  0.1969
1.0000 -1.7703  1.4338 -1.2644  1.3328 -1.3727  1.0927 -0.7082  0.3563
1.0000 -2.0396  1.9546 -1.5663  1.1368 -0.8734  0.5575 -0.1787  0.0825
1.0000 -1.6813  0.9048 -0.3997  0.2542 -0.1915  0.3392 -0.2887  0.1335
1.0000 -2.1157  1.0260  0.3798  0.1962 -0.7802  0.2204  0.1263 -0.0217
1.0000 -2.1880  1.7940 -0.8138  0.4158 -0.2620 -0.0572  0.2041 -0.0455
1.0000 -2.5457  2.6533 -1.6124  0.7142 -0.0836 -0.4030  0.4337 -0.1227

##########################################################################################
```

Fig 13 *: Codebook for the word 'zero'*

By now, we have obtained vector quantized values of the speech samples. Any speech sample can now be expressed purely in the form of a string of codebook reference numbers which can then be matched to the respective codebook. The next step will be to utilize these quantized samples to generate the model parameters for constructing a set of Hidden Markov Models and to solve Problem 3. This is accomplished using the Baum-Welch algorithm.

2.3 BAUM-WELCH ALGORITHM

To determine the parameters of a HMM it is first necessary to make a rough guess at what they might be. Once this is done, more accurate (in the maximum likelihood sense) parameters can be found by applying the so-called Baum-Welch re-estimation formulae.

Forward-Backward algorithm. Let the forward probability $\alpha_j(t)$ for some model *M* with *N* states be defined as

$$\alpha_j(t) = P(o_1, \ldots, o_t, x(t) = j | M).$$

That is, $\alpha_j(t)$ is the joint probability of observing the first *t* speech vectors and being in state *j* at time *t*. This forward probability can be efficiently calculated by the following recursion

$$\alpha_j(t) = \left[\sum_{i=2}^{N-1} \alpha_i(t-1) a_{ij} \right] b_j(o_t).$$

This recursion depends on the fact that the probability of being in state *j* at time *t* and seeing observation O_t can be deduced by summing the forward probabilities for all possible predecessor states *i* weighted by the transition probability. The initial conditions for the above recursion are

$$\alpha_1(1) = 1$$

$$\alpha_j(1) = a_{1j} b_j(o_1)$$

for $1<j<N$ and the final condition is given by

$$\alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T) a_{iN}.$$

The backward probability $\beta_i(t)$ is defined as

$$\beta_i(t) = P(o_{t+1}, \ldots, o_T | x(t) = j, M).$$

As in the forward case, this backward probability can be computed efficiently using the following recursion

$$\beta_i(t) = \sum_{j=2}^{N-1} a_{ij} b_j(o_{t+1}) \beta_j(t+1)$$

with initial conditions given by

$$\beta_i(T) = a_{iN}$$

for $1<i<N$ and final condition given by

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(o_1) \beta_j(1).$$

$$\alpha_i(t)\beta_i(t) = P(O, x(t) = j | M).$$

Procedure:

We have chosen number of states N=6. State transition is marked when there is a change in LPC coefficients pattern in time domain. Once manual segmentation is done, symbol probability matrix B is formed by normalizing the cell occupancy count. This is taken as initial symbol probability distribution in the Baum-Welch reestimation procedure. In our experiments we have used the left-right (Bakis) model, with a state transition allowed from one to itself and to the next immediate higher state only. Since the first state of Bakis model starts from state number 1, initial state probabilities are assumed as $\prod = 1.0$ and $\prod_i = 0$ for $1<i<N+1$. This parameter does not require reestimation. Other parameters are reestimated using forward-backward variables and B-W reestimation formulae. The HMM parameters $\lambda = (A,B,\prod)$ are stored and the same are shown for 'zero' and one'.

```
##########################################
     1      2      3      4      5      6
 1 0.8388 0.0000 0.0000 0.0000 0.0000 0.0000
 2 0.1612 0.8448 0.0000 0.0000 0.0000 0.0000
 3 0.0000 0.1552 0.8364 0.0000 0.0000 0.0000
 4 0.0000 0.0000 0.1636 0.7378 0.0000 0.0000
 5 0.0000 0.0000 0.0000 0.2622 0.8270 0.0000
 6 0.0000 0.0000 0.0000 0.0000 0.1730 1.0000

##########################################
```

Fig. 14 : *State Transition Probabilities (A) for the word 'One'*

```
 ##########################################
     1       2      3      4      5      6
 1 0.0356 0.0007 0.0026 0.0254 0.1144 0.0047
 2 0.0133 0.0001 0.0000 0.0132 0.2743 0.0001
 3 0.0028 0.0061 0.0063 0.2209 0.0326 0.0000
 4 0.0133 0.0001 0.0000 0.0177 0.2005 0.0000
 5 0.0442 0.0000 0.0000 0.0000 0.0000 0.0966
 6 0.2249 0.0000 0.0000 0.0000 0.0028 0.1329
 7 0.1267 0.0016 0.0012 0.0200 0.0080 0.0348
 8 0.1835 0.0554 0.0000 0.0074 0.0798 0.3145
 9 0.0007 0.0500 0.1430 0.0636 0.0022 0.0000
10 0.0000 0.1297 0.0024 0.0000 0.0020 0.0000
11 0.0949 0.0515 0.0332 0.0109 0.0000 0.0000
12 0.1846 0.0995 0.0013 0.0168 0.0000 0.0627
13 0.0037 0.0405 0.0079 0.1030 0.1200 0.0022
14 0.0377 0.0040 0.0025 0.0031 0.0202 0.0906
15 0.0000 0.1117 0.0000 0.0154 0.0000 0.0098
16 0.0165 0.1613 0.0027 0.0454 0.0570 0.0998
17 0.0000 0.0042 0.1428 0.0006 0.0036 0.0000
18 0.0000 0.0010 0.0824 0.0055 0.0000 0.0000
19 0.0096 0.0000 0.1047 0.0764 0.0045 0.0000
20 0.0000 0.0061 0.1133 0.0331 0.0073 0.0014
21 0.0000 0.0000 0.0655 0.0000 0.0000 0.0000
22 0.0000 0.0068 0.0173 0.0002 0.0000 0.0000
23 0.0000 0.0021 0.0207 0.0000 0.0000 0.0000
24 0.0000 0.0078 0.0048 0.0000 0.0000 0.0000
25 0.0000 0.0220 0.0525 0.1474 0.0516 0.0000
26 0.0000 0.0140 0.0837 0.0237 0.0090 0.0175
27 0.0078 0.0006 0.0316 0.0003 0.0024 0.1076
28 0.0001 0.0831 0.0228 0.1075 0.0056 0.0079
29 0.0000 0.0434 0.0066 0.0000 0.0000 0.0037
30 0.0000 0.0447 0.0234 0.0244 0.0013 0.0131
31 0.0000 0.0195 0.0250 0.0005 0.0000 0.0000
32 0.0000 0.0326 0.0000 0.0177 0.0007 0.0000

##########################################
```

Fig.15: *Symbol Probabilities (B) for the word 'One'*

```
##########################################

     1      2      3      4      5      6
1  0.8482 0.1518 0.0000 0.0000 0.0000 0.0000
2  0.0000 0.7929 0.2071 0.0000 0.0000 0.0000
3  0.0000 0.0000 0.7875 0.2125 0.0000 0.0000
4  0.0000 0.0000 0.0000 0.8745 0.1255 0.0000
5  0.0000 0.0000 0.0000 0.0000 0.8742 0.1258
6  0.0000 0.0000 0.0000 0.0000 0.0000 1.0000

##########################################
```

Fig. 16: *State Transition Probabilities (A) for zero*

```
##########################################
     1       2      3      4      5      6
1  0.0000 0.0021 0.0042 0.1607 0.0083 0.0000
2  0.0000 0.0000 0.0000 0.0036 0.1687 0.0000
3  0.0000 0.0000 0.0000 0.0648 0.0007 0.0000
4  0.0000 0.0000 0.0000 0.0530 0.0136 0.0000
5  0.0000 0.0000 0.0000 0.1134 0.0148 0.0000
6  0.0000 0.0000 0.0000 0.0629 0.0015 0.0000
7  0.0000 0.0000 0.0000 0.0426 0.0000 0.0000
8  0.0000 0.0000 0.0000 0.0200 0.0000 0.0000
9  0.0000 0.0000 0.0000 0.0373 0.0000 0.0000
10 0.0000 0.0000 0.0000 0.0267 0.0000 0.0000
11 0.0000 0.0000 0.0000 0.0307 0.0000 0.0000
12 0.0000 0.0000 0.0000 0.0107 0.0000 0.0000
13 0.0000 0.0000 0.0000 0.0613 0.0000 0.0000
14 0.0000 0.0000 0.0000 0.0187 0.0000 0.0000
15 0.0000 0.0000 0.0000 0.0360 0.0000 0.0000
16 0.0000 0.0000 0.0000 0.0147 0.0000 0.0000
17 0.1329 0.0968 0.0000 0.0000 0.0000 0.0000
18 0.1336 0.0028 0.0000 0.0000 0.0000 0.0000
19 0.1145 0.0000 0.0000 0.0000 0.0000 0.0075
20 0.1293 0.0160 0.0000 0.0000 0.0000 0.0174
21 0.1168 0.0036 0.0000 0.0000 0.0000 0.0174
22 0.0089 0.0000 0.0000 0.0000 0.0000 0.1765
23 0.2015 0.2204 0.0000 0.0000 0.0000 0.0323
24 0.1297 0.3925 0.0000 0.0000 0.0057 0.2261
25 0.0016 0.0008 0.4776 0.0000 0.0000 0.0000
26 0.0000 0.1690 0.0072 0.0000 0.2472 0.0390
27 0.0000 0.0089 0.2539 0.0097 0.1774 0.0000
28 0.0018 0.0000 0.0666 0.1074 0.0335 0.0029
29 0.0043 0.0074 0.0415 0.0595 0.2011 0.0453
30 0.0000 0.0000 0.0000 0.0526 0.0467 0.0062
31 0.0197 0.0797 0.1488 0.0136 0.0468 0.0302
32 0.0054 0.0000 0.0000 0.0003 0.0340 0.3992

##########################################
```

Fig. 17: *Symbol Probabilities (B) for zero*

Once the training has been accomplished, we move on to the recognition phase using the Viterbi algorithm.

2.4 VITERBI ALGORITHM FOR MAXIMUM LIKELIHOOD COMPUTATION

To find the single best state sequence, $Q = \{q_1\ q_2, \ldots q_r\}$ **,** for the given observation sequence $O = \{O_1, O_2, O_3, \ldots O_T)$ we need to define the quantity

$$\delta_t(i) = \max_{q_1, q_2, \cdots, q_{t-1}} P[q_1\ q_2 \cdots q_t = i,\ O_1\ O_2 \cdots O_t | \lambda]$$

i.e., $\delta_t(i)$ is the best score (highest probability) along a single path, at time *t*, which accounts for the first t observations and ends in state S_i**.** By induction we have

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(O_{t+1}).$$

To actually retrieve the state sequence, we need to keep track of the argument which maximized the above equation for each t and j. We do this via the array $\psi_t(j)$. The complete procedure for finding the best state sequence can now be stated as follows:

1) Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \qquad 1 \le i \le N$$

$$\psi_1(i) = 0.$$

2) Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \qquad 2 \le t \le T$$

$$1 \le j \le N$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \qquad 2 \le t \le T$$

$$1 \le j \le N.$$

3) Termination:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$

$$q_T^* = \operatorname*{argmax}_{1 \le i \le N} [\delta_T(i)].$$

4) Path (state sequence) backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T - 1, T - 2, \cdots, 1.$$

SOFTWARE TEST RESULTS

The following tests were performed by taking 20 speech samples spoken by five different speakers for each of the words “Zero”, “One”, “Two”, “Three”, “Four”.

Recognition Accuracy (%)

Word spoken as	Word Recognized as “Zero”	“One”	“Two”	“Three”	“Four”
“Zero”	98	2	0	0	0
“One”	13	83	1	0	3
“Two”	11	3	65	3	18
“Three”	0	2	0	75	23
“Four”	3	1	14	2	80

3. IMPLEMENTATION OF THE VITERBI ALGORITHM ON A FIELD PROGRAMMABLE GATE ARRAY

After achieving a high degree of accuracy using MATLAB simulation of the Isolated Speech Recognition system, we now turn our attention to increasing the speed at which the computation is processed. This is possible by implementing the Viterbi algorithm for maximum probability computation on a hardware platform. This involves design of the VHDL code, generating an RTL logic circuit and finally, conversion to a bitstream sequence which can then be ported on to an FPGA.

3.1 A BRIEF INTRODUCTION TO VHDL

VHDL is used mainly for the development of Application Specific Integrated Circuits (ASICs). Tools for the automatic transformation of VHDL code into a gate-level netlist were developed already at an early point of time. This transformation is called synthesis and is an integral part of current design flows. The development of VHDL models starts with their specification which covers functional aspects and the timing behavior. Sometimes a behavioral VHDL model is derived from there, yet synthesizable code is frequently requested right from the beginning. VHDL code can be simulated and checked for the proper functionality.

If the model shows the desired behavior, the VHDL description will be synthesized. A synthesis tool selects the appropriate gates and flip-flops from the specified ASIC library in order to reproduce the functional description.

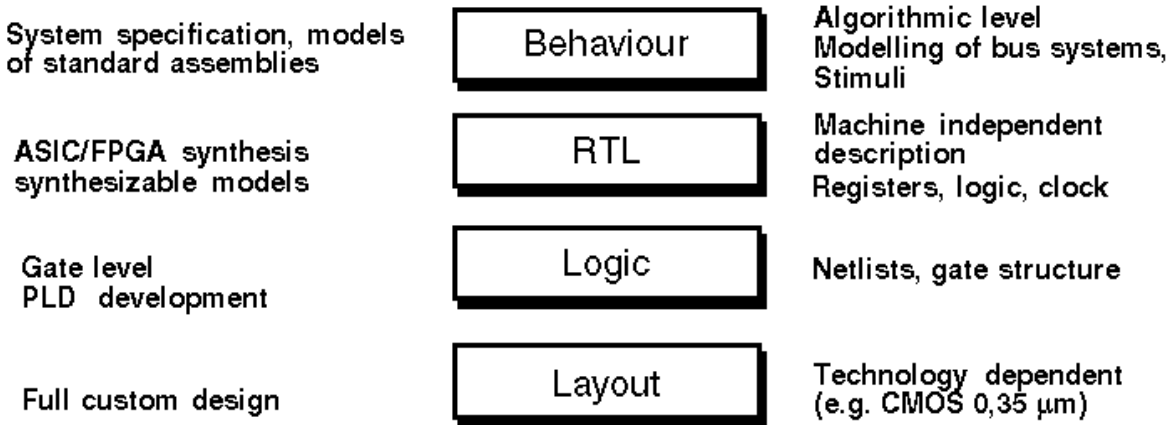


Fig. 18: *VHDL Abstraction Levels*

In the behavior level, complete systems can be modeled. Bus systems or complex algorithms are described without considering synthesizability. The stimuli for simulation of RTL models are described in the behavior level, for example. Stimuli are signal values of the input ports of the model and are described in the testbench, sometimes called validation bench.

The designer has to take great care to find a consistent set of input stimuli that do not contradict the specification. The responses of the model have to be compared with the expected values which, in the simplest case, can be done with the help of a waveform diagram that shows the simulated signal values.

On the RT level, the system is described in terms of registers and logic that calculates the next value of the storage elements. It is possible to split the code into two blocks (cf. process statement) that contain either purely combinational logic or registers. The registers are connected to the clock signal and provide for synchronous behavior. In practice, the strict separation of Flip Flops from combinational logic is often annulated and clocked processes describe the registers and the corresponding update functions.

The gate netlist is generated from the RT description with the help of a synthesis tool. For this task, a cell library for the target technology which holds the information about all available gates and their parameters (fan-in, fan-out, delay) is needed.

Based upon this gate netlist the circuit layout is generated. The resulting wire lengths can be converted into propagation delays which can be fed back into the gate level model (back annotation). This allows for thorough timing simulations without the need for additional simulator software.

3.2 A BRIEF INTRODUCTION TO FIELD PROGRAMMABLE GATE ARRAYS

Field **P**rogrammable **G**ate **A**rrays are two dimensional arrays of logic blocks and flip-flops with a electrically programmable interconnections between logic blocks.

In an FPGA logic blocks are implemented using multiple level low fan-in gates, which gives it a more compact design compared to an implementation with two-level AND-OR logic. FPGA provides its user a way to configure:

1. The intersection between the logic blocks and
2. The function of each logic block.

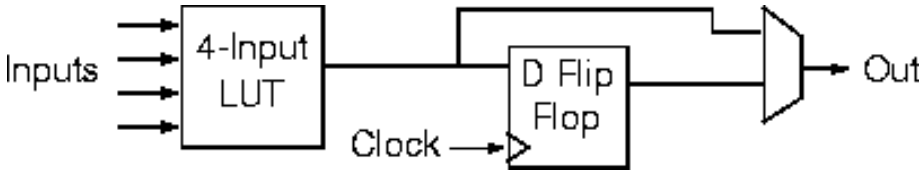
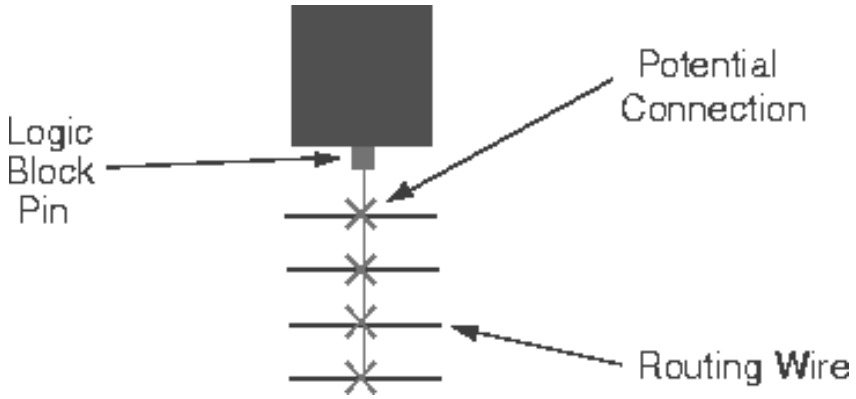


Fig 19: *Logic Block*

Logic block of an FPGA can be configured in such a way that it can provide functionality as simple as that of transistor or as complex as that of a microprocessor. It can used to implement different combinations of combinational and sequential logic functions. Logic blocks of an FPGA can be implemented by any of the following:

1. Transistor pairs
2. combinational gates like basic NAND gates or XOR gates
3. n-input Lookup tables
4. Multiplexers
5. Wide fan-in AND-OR structure.



Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it. The figure should make the situation clear.

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

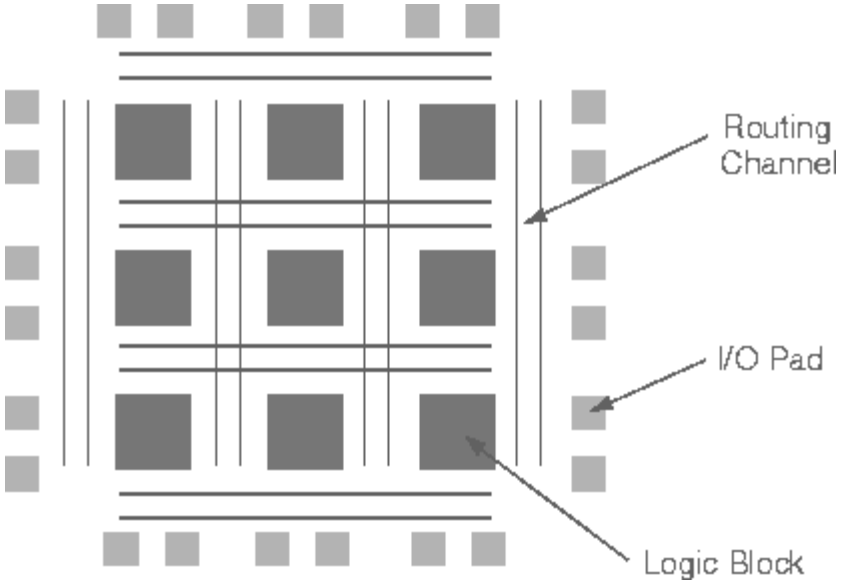


Fig. 20: *FPGA Structure*

Routing in FPGAs consists of wire segments of varying lengths which can be interconnected via electrically programmable switches. Density of logic block used in an FPGA depends on length and number of wire segments used for routing. Number of segments used for interconnection typically is a tradeoff between density of logic blocks used and amount of area used up for routing.

The ability to reconfigure functionality to be implemented on a chip gives a unique advantage to designer who designs his system on an FPGA It reduces the time to market and significantly reduces the cost of production.

3.3 FPGA IMPLEMENTATION OF THE VITERBI ALGORITHM

The Viterbi algorithm was modified suitably to be implemented on an FPGA. Since the FPGA performs parallel computations like addition and comparisons much faster, the algorithm was taken in the log domain. This removes any multiplications from the calculations thus making the VHDL code easier to synthesize.

1) Initialization

$$A^* = log(A)$$

$$B^* = log(B)$$

$$\prod^* = log(\prod)$$

$$\delta^*_1(i) = \prod^*_i + B^*_i(O_1) \qquad 1 \leq i \leq N$$

2) Recursion

$$\delta^*_t(j) = max\ [\ \delta^*_{t-1}(i) + A^*_{ij}\ ] + B^*_j(O_t) \qquad 2 \leq t \leq T$$

$$1 \leq j \leq N$$

$$1 \leq i \leq N$$

3) Termination

$$P^* = max\ [\ \delta^*_T(i)] \qquad 1 \leq i \leq N$$

3.4 ISSUES FACED DURING HARDWARE DESIGN AND SYNTHESIS

Besides the fixed synthesis constraints set by the target technology and the tool capabilities, "soft" constraints that are imposed by the designer have to be considered as well. Maximum operating speed and required hardware resources are usually the main targets for netlist optimization. This is possible either on a purely abstract mathematical model or by different mappings of the boolean functions on the available technology cells. Due to the complexity, the optimization phase requires quite a lot of iterations before the software reports its final result.

Essential Information for Synthesis

- Load values
- Path delays
- Driver strengths
- Timing
- Operating conditions

Problems with Synthesis Tools

- Timing issues
    - layout information is missing during the synthesis process
    - clock tree must be generated afterwards
- Complex clocking schemes
  (inverted clocks, multiple clocks, gated clocks)
- Memory
    - synthesis tools are not able to replace register arrays with memory macro cells

- Macro cells
    - no standardized way for instantiation of existing technology macro cells
- IO-pads
    - ASIC-libraries have several different IO-pads
    - selection by hand, either within the synthesis tool or in the top level entity

The VHDL coding style itself has a rather big impact on the synthesis result. Therefore it is necessary to keep this in mind even if the model is to be synthesized at the last step of the development cycle.

HARDWARE SIMULATION TEST RESULTS

The VHDL simulation was carried out in the ModelSim software simulator. The program requires 32 clock cycles to compute. At a clock frequency of 25 MHz, we get a clock duration of 40 ns. The synthesis was carried out in Xilinx Project Navigator and the synthesis reports as well as the RTL schematics were generated.

Implementation of the Viterbi Algorithm for maximum probability computation using RTL results in a lower recognition time, as opposed to a standard processor setting.

Comparing the average time taken for a word to be recognized in Software and Hardware mode –

MATLAB Recognition Time: 79 us

VHDL Recognition Time: 13 us

The VHDL model shows a reduction in the time taken to recognize by a factor of 6 over the MATLAB model.

CONCLUSIONS

1. Speech signal which changes its characteristics with time can be modeled stochastically in the form of HMM parameters and codebook.

2. Usage of an FPGA over MATLAB has enhanced the overall recognition time by 6 times.

3. Choice of number of states N=6 is satisfactory and there is no simple relationship existing between recognition accuracy, the number of sounds in the word and the number of states needed in an HMM.

4. Single model per word should be adequate and effects of different random starts (initial model) are also insignificant.

The above conclusions point out that the FPGA approach is very attractive for speech recognition and is inexpensive in terms of storage and speed.

Following are the suggestions of future work in this field.

1. Recognition of isolated digits using HMM with continuous mixture densities gives improved recognition accuracy.

2. Porting of the entire pre-processing stage (Windowing + LPC + VQ) to the FPGA would result in a stand alone ASIC for speech recognition capable of running independent of any processor.

3. Removing redundancy in calculations in HMM as well as VHDL would further reduce recognition times.

REFERENCES

[1] Goslin, G. R. "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices," Proceedings of the DSPX 1995 Technical Proceedings pp. 595-604, 12JAN95.

[2] "The programmable Logic Data Book," Xilinx, Inc. Second Edition, 1994

[3] AL Gorin, G Riccardi and JH Wright, "How may I help you?", *Speech Communication* 23, (1997) pp 113-127.

[4] Rabiner, L.R., "A tutorial on Hidden Markov Models and selected applications in speech recognition," *Proc. IEEE*, **77**, No.2, 1989, pp.257-286.

[5] The various voice and speech toolboxes available at www.mathtools.net

[6] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE* Trans. Informat. Theory, vol. IT-13, pp. 260-269, Apr. 1967.

[7] S. E. Levinson, "Structural methods in automatic speech recognition," Proc. *IEEE*, vol. 73, no. 11, pp. 1625-1650, Nov. 1985.

[8] S. E. Levinson, "Continuously variable duration hidden Mar- kov models for automatic speech recognition," Computer, Speech and Language, vol. 1, no. 1, pp. 29-45, Mar. 1986.

[9] The Xilinx Spartan II Product Manual can be downloaded from www.xilinx.com.

[10] A lot of other resources can be located at www.speech.cs.cmu.edu.

[11] Junqua, Jean-Claude, "Robust Speech Recognition in Embedded Systems and PC Applications", Kluwer Academic Publishers

[12] Melnikoff, S.J., Quigley, S.F. & Russell, M.J.,"Implementing a hidden Markov model speech recognition system in programmable logic," *FPL 2001, LNCS #2147*, 2001, pp.81-90.

[13] L. R. Rabiner, B. H. Juang, S. E. Levinson, and M. M. Sondhi, "Recognition of isolated digits using hidden Markov models with continuous mixture densities," AT&T Tech. *J.*, vol. **64,** no. 6, pp. **1211-1222,** July-Aug. **1986.**

[14] Information on VHDL can be found on www.eda.org and www.vhdlonline.de

APPENDIX – A

MATLAB CODE FOR SPEECH RECOGNITION

1. FrameBlock.m

```
function Frames = FrameBlock(input, N, M)
% this function cuts an input vector into frames with overlap

input = input';
nbFrames = ceil((length(input)-N)/M);
     for i = 0:nbFrames-1

        temp = input(i*M+1:i*M+N);
        Frames(i+1,1:N) = temp;

        i=i+1;
     end
%last Frame:

temp = zeros(1,N);
lastLength = length(input)-(nbFrames-1)*M -N +N-M;

temp(1:lastLength) = input(nbFrames*M+1:(nbFrames*M +1 + lastLength-
1));
Frames(nbFrames+1, 1:N) = temp;
```

2. Windowing.m

```
function Windows = Windowing(frames)
% this function applies the Hamming Window the each frame of the input
Matrix
% each line is a frame

frameSize = size(frames);
nbFrames = frameSize(1);
nbSamples = frameSize(2);

%Hamming
w = hamming(nbSamples);

     for i = 1:nbFrames
          temp = frames(i,1:nbSamples);
          Windows(i, 1:nbSamples) = w'.*temp;
end
```

3. filename.m

```
function [filename] = filename(i,j,number);
filename = 'C:\Documents and Settings\Sandy\My Documents\Training
Data\Speaker ';
speaker_index = num2str(i);
data_index = num2str(j);

filename = strcat([filename speaker_index '\' number '_' data_index]);
```

4. lpc_coeff.m

```
function [L] = lpc_coeff(filename)
a = wavread(filename);
f = FrameBlock(a,420,180);
w = Windowing(f);
w = w';
L = lpc(w,8);
```

5. disteu.m

```
function d = disteu(x, y)
% DISTEU Pairwise Euclidean distances between columns of two matrices
%
% Input:
%       x, y:   Two matrices whose each column is an a vector data.
%
% Output:
%       d:      Element d(i,j) will be the Euclidean distance between
two
%               column vectors X(:,i) and Y(:,j)
%
% Note:
%       The Euclidean distance D between two vectors X and Y is:
%       D = sum((x-y).^2).^0.5

[M, N] = size(x);
[M2, P] = size(y);

if (M ~= M2)
    error('Matrix dimensions do not match.')
end

d = zeros(N, P);

if (N < P)
    copies = zeros(1,P);
    for n = 1:N
        d(n,:) = sum((x(:, n+copies) - y) .^2, 1);
    end
else
    copies = zeros(1,N);
    for p = 1:P
        d(:,p) = sum((x - y(:, p+copies)) .^2, 1)';
    end
end

d = d.^0.5;
```

6. SplitCodeword.m

```
function [c, ind] = SplitCodeword(c, d, indNew)

% this function  splits one input codeword into two
% c contains the codeword to split
% d contains all the samples
% indNew indicates which samples of d to take for the calculation of
the new centroids.

%returns:
```

```
% c contains the two new codewords
% ind is a vector of 0, 1 and 2. ind(i) indicates which of the two
codewords is closer to the sample i.
% if ind(i) = 0, the sample is not in the subspace of c.

e = 0.001;
dimY = 9;

% reduce d to the samples that should be considered for the codeword c
d = d(:, find(indNew ~= 0));


% split centeroid into two
for i = 1:dimY

tmp1(i) = c(i) + e;
tmp2(i) = c(i) - e;

end;

c = [tmp1; tmp2]';

%Nearest neighbor search
z = disteu(d,c);

% z contains the euclidean distance of each vector in d to the two
vectors tmp1 and tmp2 (integrated in c)
% one row per sample and a column for tmp1 and one for tmp2


[m, ind] = min(z, [], 2);
% m is that value of minimum of each row of z. ind contains the index
of this minimum,
% i.e. which of the two children of the old  centroids
% is nearer to the corresponding sample.

ctmp = c;

for j =1:2

   c(:,j) = mean(d(:, find(ind == j)), 2);
end;
% for each subspace of d calculate the new centroid


%loop till c doesn't change more than e
% compare the two new centroids to the children of the old centroid

diff = max( max(abs(c-ctmp)));
% diff is the maximum distance of movment of any centroid in any
direction.

while diff > e
      %Nearest neighbor search
      % repeat the same till both of the centroids move less than e in
any direction
```

```
      z = disteu(d,c);
      [m, ind] = min(z, [], 2);

      ctmp = c;

      for j =1:2
         c(:,j) = mean(d(:, find(ind == j)), 2);
      end;


      diff = max( max(abs(c-ctmp)));
end;

% reconstruct ind to return
% write the next centroid into ind only for the samples concearning c

ind= ind';

count = 1;

si = size(indNew);
sInd = si(2);
for m = 1:sInd

      if indNew(m) ~= 0
                  indNew(m) = ind(count);
                    count = count +1;
         end;
end;

ind = indNew;
```

7. vqlbg.m

```
function c = vqlbg(d, k)

% VQLBG Vector quantization using the Linde-Buzo-Gray algorithm

%

% Inputs:

%       d contains training data vectors (one per column)

%       k is number of centroids required

%

% Outputs:

%       c contains the result VQ codebook (k columns, one for each
centroids)

e = 0.001;
dimY = 9;

sx = size(d);
dimX = sx(2);
```

```
% design a 1-vector codebook containing the centeroid of d.

c1 = sum(d,2);
c2 = size(d,2);
c = c1/c2;

% at the beginning the whole space of d is considered.
dSub = d;

% calculate the number of iterations needed to obtain at least k
centroids (this number is always a power of 2)

it = ceil(log2(k));


%ind indicates for each sample which is the nearest centroid that has
already been created.
ind = ones(1,dimX);

for i = 1:it   %first for loop
    % calcl CurrentCodebookLength
    sz = size(c);
    ccl = sz(2);
    %for each existing codeword in the old codebook cOld make two
codewords and add them to the codebook c
    cOld = c;
    indM = ind;
    for j = 1:ccl   %second for loop
        %calculate the subspace out of which we want to calculate two
new centroids
        indNew = ind;
        for m = 1:dimX
            if indNew(m) ~= j
                  indNew(m) = 0;
            end;
        end;
        [pair, indtmp] = SplitCodeword(cOld(:,j), d, indNew);
    %update ind
        for m = 1:dimX
            if indtmp(m) ~= 0
              %  ind(m) = j*2+indtmp(m)-2;
            indM(m) = j*2+indtmp(m)-2;
            end;
        end;
        if j==1
            c = pair;
        else
            c = [c, pair];
        end;
    end;    %end of second for loop
    ind = indM;
end;  %end of first for loop
```

8. uniform_seg.m

```
function [state_seq_rc, state_seq_singlerow] = uniform_seg()

state_seq_rc = zeros(100,32);
for i=1:100
    state_seq_rc(i,1:6) = 1;
    state_seq_rc(i,7:12) = 2;
    state_seq_rc(i,13:17) = 3;
    state_seq_rc(i,18:22) = 4;
    state_seq_rc(i,23:27) = 5;
    state_seq_rc(i,28:32) = 6;
end
for i=1:100
    k6 = 32*i-31;
    k7 = 32*i;
    state_seq_singlerow(1,k6:k7) = state_seq_rc(i,:);
end
```

9. vector_quantize.m

```
function [C, data, I, obs_seq_lpcform] = vector_quantize(cbk, lpc_data)

data = zeros(100,32);
obs_seq_lpcform = zeros(9,3200);
D = disteu(cbk, lpc_data);
[C, I] = min(D);
for i=1:100
    k1 = 32*i-31;
    k2 = 32*i;
    data(i,:) = I(k1:k2);
end
for i=1:3200
    k3 = I(i);
    obs_seq_lpcform(:,i) = cbk(:,k3);
end
```

10. train_manualseg.m

```
function [A3, B3, C] = train_manualseg(number)

Q = 6;
O = 32;
D = zeros(9,3200);
c = 0;
for i = [1:5]
    for j = [1:20]
        c = c+1;
        file_path = filename(i,j,number);
        L = lpc_coeff(file_path);
        L = L';
        k1 = 32*(c-1) + 1;
        k2 = 32*c;
        D(1:9, k1:k2) = L(1:9, 1:32);
        out_file_path = strcat([file_path '.txt']);
        fid = fopen(out_file_path,'wt');
        fprintf(fid,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',L);
        fclose(fid);
    end
```

```
end
C = vqlbg(D,32);
out_file_path = strcat(['C:\Documents and Settings\Sandy\My
Documents\Training Data\codebook_' number '.txt']);
fid = fopen(out_file_path,'wt');
fprintf(fid,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f %6.4f
%6.4f\n',C);
fclose(fid);
[min_value, obs_seq_rc, obs_seq_singlerow, obs_seq_lpcform] =
vector_quantize(C, D);

[states_seq_rc, states_seq_singlerow] = uniform_seg();

[A2,B2] = hmmestimate(obs_seq_singlerow, states_seq_singlerow);
% replace zero entries in B by 0.0001
A2(6,1) = 0;
A2(6,6) = 1;
[R0,C0] = find(B2==0);
for i=1:size(R0,1)
    B2(R0(i),C0(i)) = 0.0001;
end

[A3,B3] = hmmtrain(obs_seq_rc, A2, B2,'TOLERANCE', 0.001,
'MAXITERATIONS', 100);

[R0,C0] = find(B2==0);
for i=1:size(R0,1)
    B2(R0(i),C0(i)) = 0.0001;
end

out_file_path = strcat(['C:\Documents and Settings\Sandy\My
Documents\Training Data\A_m_' number '.txt']);
fid = fopen(out_file_path,'wt');
fprintf(fid,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f\n',A3);
fclose(fid);
out_file_path = strcat(['C:\Documents and Settings\Sandy\My
Documents\Training Data\B_m_' number '.txt']);
fid = fopen(out_file_path,'wt');
fprintf(fid,'%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f\n',B3);
fclose(fid);
```

11. recog_loop.m

```
function [c] = recog_loop(A0, B0, C0, A1, B1, C1, A2, B2, C2, A3, B3,
C3, A4, B4, C4)

c = zeros(5,5);
for i=1:5
    for j=1:20
        file_path = filename(i,j,'zero');
        ll0 = recog(file_path, A0, B0, C0);
        ll1 = recog(file_path, A1, B1, C1);
        ll2 = recog(file_path, A2, B2, C2);
        ll3 = recog(file_path, A3, B3, C3);
        ll4 = recog(file_path, A4, B4, C4);
        k = max([ll0,ll1,ll2,ll3,ll4]);
        if(ll0==k)
            c(1,1) = c(1,1)+1;
```

```
        end
        if(ll1==k)
            c(1,2) = c(1,2)+1;
        end
        if(ll2==k)
            c(1,3) = c(1,3)+1;
        end
        if(ll3==k)
            c(1,4) = c(1,4)+1;
        end
        if(ll4==k)
            c(1,5) = c(1,5)+1;
        end

    end
end
for i=1:5
    for j=1:20
        file_path = filename(i,j,'one');
        ll0 = recog(file_path, A0, B0, C0);
        ll1 = recog(file_path, A1, B1, C1);
        ll2 = recog(file_path, A2, B2, C2);
        ll3 = recog(file_path, A3, B3, C3);
        ll4 = recog(file_path, A4, B4, C4);
        k = max([ll0,ll1,ll2,ll3,ll4]);
        if(ll0==k)
            c(2,1) = c(2,1)+1;
        end
        if(ll1==k)
            c(2,2) = c(2,2)+1;
        end
        if(ll2==k)
            c(2,3) = c(2,3)+1;
        end
        if(ll3==k)
            c(2,4) = c(2,4)+1;
        end
        if(ll4==k)
            c(2,5) = c(2,5)+1;
      end
end
end
for i=1:5
    for j=1:20
        file_path = filename(i,j,'two');
        ll0 = recog(file_path, A0, B0, C0);
        ll1 = recog(file_path, A1, B1, C1);
        ll2 = recog(file_path, A2, B2, C2);
        ll3 = recog(file_path, A3, B3, C3);
        ll4 = recog(file_path, A4, B4, C4);
        k = max([ll0,ll1,ll2,ll3,ll4]);
        if(ll0==k)
            c(3,1) = c(3,1)+1;
        end
        if(ll1==k)
            c(3,2) = c(3,2)+1;
        end
```

```
        if(ll2==k)
            c(3,3) = c(3,3)+1;
        end
        if(ll3==k)
            c(3,4) = c(3,4)+1;
        end
        if(ll4==k)
            c(3,5) = c(3,5)+1;
        end

    end
end
for i=1:5
    for j=1:20
        file_path = filename(i,j,'three');
        ll0 = recog(file_path, A0, B0, C0);
        ll1 = recog(file_path, A1, B1, C1);
        ll2 = recog(file_path, A2, B2, C2);
        ll3 = recog(file_path, A3, B3, C3);
        ll4 = recog(file_path, A4, B4, C4);
        k = max([ll0,ll1,ll2,ll3,ll4]);
        if(ll0==k)
            c(4,1) = c(4,1)+1;
        end
        if(ll1==k)
            c(4,2) = c(4,2)+1;
        end
        if(ll2==k)
            c(4,3) = c(4,3)+1;
        end
        if(ll3==k)
            c(4,4) = c(4,4)+1;
        end
        if(ll4==k)
            c(4,5) = c(4,5)+1;
        end

    end
end
for i=1:5
    for j=1:20
        file_path = filename(i,j,'four');
        ll0 = recog(file_path, A0, B0, C0);
        ll1 = recog(file_path, A1, B1, C1);
        ll2 = recog(file_path, A2, B2, C2);
        ll3 = recog(file_path, A3, B3, C3);
        ll4 = recog(file_path, A4, B4, C4);
        k = max([ll0,ll1,ll2,ll3,ll4]);
        if(ll0==k)
            c(5,1) = c(5,1)+1;
        end
        if(ll1==k)
            c(5,2) = c(5,2)+1;
        end
        if(ll2==k)
            c(5,3) = c(5,3)+1;
        end
```

```
        if(ll3==k)
            c(5,4) = c(5,4)+1;
        end
        if(ll4==k)
            c(5,5) = c(5,5)+1;
        end

    end
end
```

APPENDIX – B
VHDL CODE FOR THE VITERBI ALGORITHM

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;


LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_SIM IS
    PORT(CLOCK : IN STD_LOGIC;
         RECOG_NO : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
END VITERBI_SIM;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_ALGO0 IS
    PORT(CLOCK   : IN STD_LOGIC;
              ENABLE   : OUT STD_LOGIC;
              ADDRESS  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
              DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              DATA_READY : IN STD_LOGIC;
              MAX_ENABLE : OUT STD_LOGIC;
              LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
         );
END VITERBI_ALGO0;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_ALGO1 IS
    PORT(CLOCK   : IN STD_LOGIC;
              ENABLE   : OUT STD_LOGIC;
              ADDRESS  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
              DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
              DATA_READY : IN STD_LOGIC;
              MAX_ENABLE : OUT STD_LOGIC;
              LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
         );
END VITERBI_ALGO1;

LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_ALGO2 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : OUT STD_LOGIC;
             ADDRESS  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             DATA_READY : IN STD_LOGIC;
             MAX_ENABLE : OUT STD_LOGIC;
             LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END VITERBI_ALGO2;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_ALGO3 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : OUT STD_LOGIC;
             ADDRESS  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             DATA_READY : IN STD_LOGIC;
             MAX_ENABLE : OUT STD_LOGIC;
             LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END VITERBI_ALGO3;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY VITERBI_ALGO4 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : OUT STD_LOGIC;
             ADDRESS  : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             DATA_READY : IN STD_LOGIC;
             MAX_ENABLE : OUT STD_LOGIC;
             LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END VITERBI_ALGO4;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM_MODULE0 IS
```

```
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : IN STD_LOGIC;
             ADDRESS  : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_READY : OUT STD_LOGIC;
             DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END ROM_MODULE0;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM_MODULE1 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : IN STD_LOGIC;
             ADDRESS  : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_READY : OUT STD_LOGIC;
             DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END ROM_MODULE1;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM_MODULE2 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : IN STD_LOGIC;
             ADDRESS  : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_READY : OUT STD_LOGIC;
             DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END ROM_MODULE2;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM_MODULE3 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : IN STD_LOGIC;
             ADDRESS  : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_READY : OUT STD_LOGIC;
             DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END ROM_MODULE3;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

```
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROM_MODULE4 IS
    PORT(CLOCK   : IN STD_LOGIC;
             ENABLE   : IN STD_LOGIC;
             ADDRESS  : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
             DATA_READY : OUT STD_LOGIC;
             DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END ROM_MODULE4;

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY MAX_MODULE IS
    PORT(CLOCK : IN STD_LOGIC;
         MAX_ENABLE0 : IN STD_LOGIC;
         MAX_ENABLE1 : IN STD_LOGIC;
                     MAX_ENABLE2 : IN STD_LOGIC;
                     MAX_ENABLE3 : IN STD_LOGIC;
                     MAX_ENABLE4 : IN STD_LOGIC;
         LOG_LIK0 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         LOG_LIK1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                     LOG_LIK2 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                     LOG_LIK3 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                     LOG_LIK4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         RECOG_NO : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
END MAX_MODULE;

ARCHITECTURE VITERBI_SIM OF VITERBI_SIM IS
    COMPONENT VITERBI_ALGO0
        PORT(CLOCK   : IN STD_LOGIC;
                 ENABLE      : OUT STD_LOGIC;
                 ADDRESS     : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
                 DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                 DATA_READY : IN STD_LOGIC;
                 MAX_ENABLE : OUT STD_LOGIC;
                 LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
            );
    END COMPONENT;
    COMPONENT VITERBI_ALGO1
        PORT(CLOCK   : IN STD_LOGIC;
                 ENABLE      : OUT STD_LOGIC;
                 ADDRESS     : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
                 DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                 DATA_READY : IN STD_LOGIC;
                 MAX_ENABLE : OUT STD_LOGIC;
                 LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
            );
    END COMPONENT;
         COMPONENT VITERBI_ALGO2
        PORT(CLOCK   : IN STD_LOGIC;
                 ENABLE      : OUT STD_LOGIC;
```

```
            ADDRESS     : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            DATA_READY : IN STD_LOGIC;
            MAX_ENABLE : OUT STD_LOGIC;
            LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END COMPONENT;
    COMPONENT VITERBI_ALGO3
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : OUT STD_LOGIC;
            ADDRESS     : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            DATA_READY : IN STD_LOGIC;
            MAX_ENABLE : OUT STD_LOGIC;
            LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END COMPONENT;
    COMPONENT VITERBI_ALGO4
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : OUT STD_LOGIC;
            ADDRESS     : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_IN: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            DATA_READY : IN STD_LOGIC;
            MAX_ENABLE : OUT STD_LOGIC;
            LOG_LIK : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END COMPONENT;
COMPONENT ROM_MODULE0
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : IN STD_LOGIC;
            ADDRESS     : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_READY : OUT STD_LOGIC;
            DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;
COMPONENT ROM_MODULE1
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : IN STD_LOGIC;
            ADDRESS     : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_READY : OUT STD_LOGIC;
            DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;
    COMPONENT ROM_MODULE2
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : IN STD_LOGIC;
            ADDRESS     : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_READY : OUT STD_LOGIC;
            DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;
    COMPONENT ROM_MODULE3
    PORT(CLOCK   : IN STD_LOGIC;
            ENABLE      : IN STD_LOGIC;
            ADDRESS     : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
            DATA_READY : OUT STD_LOGIC;
            DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
```

```
        );
END COMPONENT;
     COMPONENT ROM_MODULE4
    PORT(CLOCK   : IN STD_LOGIC;
              ENABLE      : IN STD_LOGIC;
              ADDRESS     : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
              DATA_READY : OUT STD_LOGIC;
              DATA_OUT: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
        );
END COMPONENT;
COMPONENT MAX_MODULE
    PORT(CLOCK : IN STD_LOGIC;
         MAX_ENABLE0 : IN STD_LOGIC;
         MAX_ENABLE1 : IN STD_LOGIC;
                         MAX_ENABLE2 : IN STD_LOGIC;
                         MAX_ENABLE3 : IN STD_LOGIC;
                         MAX_ENABLE4 : IN STD_LOGIC;
         LOG_LIK0 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         LOG_LIK1 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                         LOG_LIK2 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                         LOG_LIK3 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                         LOG_LIK4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
         RECOG_NO : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
         );
END COMPONENT;

SIGNAL W_ENABLE0 : STD_LOGIC;
SIGNAL W_ADDRESS0 : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL W_DATA0 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL W_ENABLE1 : STD_LOGIC;
SIGNAL W_ADDRESS1 : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL W_DATA1 : STD_LOGIC_VECTOR(7 DOWNTO 0);
     SIGNAL W_ENABLE2 : STD_LOGIC;
SIGNAL W_ADDRESS2 : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL W_DATA2 : STD_LOGIC_VECTOR(7 DOWNTO 0);
     SIGNAL W_ENABLE3 : STD_LOGIC;
SIGNAL W_ADDRESS3 : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL W_DATA3 : STD_LOGIC_VECTOR(7 DOWNTO 0);
     SIGNAL W_ENABLE4 : STD_LOGIC;
SIGNAL W_ADDRESS4 : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL W_DATA4 : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL W_LOG_LIK0 : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL W_LOG_LIK1 : STD_LOGIC_VECTOR(31 DOWNTO 0);
     SIGNAL W_LOG_LIK2 : STD_LOGIC_VECTOR(31 DOWNTO 0);
     SIGNAL W_LOG_LIK3 : STD_LOGIC_VECTOR(31 DOWNTO 0);
     SIGNAL W_LOG_LIK4 : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL W_DATA_READY0 : STD_LOGIC;
SIGNAL W_DATA_READY1 : STD_LOGIC;
     SIGNAL W_DATA_READY2 : STD_LOGIC;
     SIGNAL W_DATA_READY3 : STD_LOGIC;
     SIGNAL W_DATA_READY4 : STD_LOGIC;
SIGNAL W_MAX_ENABLE0 : STD_LOGIC;
SIGNAL W_MAX_ENABLE1 : STD_LOGIC;
     SIGNAL W_MAX_ENABLE2 : STD_LOGIC;
     SIGNAL W_MAX_ENABLE3 : STD_LOGIC;
     SIGNAL W_MAX_ENABLE4 : STD_LOGIC;
```

```
    BEGIN
    VA0 : VITERBI_ALGO0
    PORT MAP(CLOCK, W_ENABLE0, W_ADDRESS0, W_DATA0, W_DATA_READY0, W_MAX_ENABLE0,
W_LOG_LIK0);
    VA1 : VITERBI_ALGO1
    PORT MAP(CLOCK, W_ENABLE1, W_ADDRESS1, W_DATA1, W_DATA_READY1, W_MAX_ENABLE1,
W_LOG_LIK1);
    VA2 : VITERBI_ALGO2
    PORT MAP(CLOCK, W_ENABLE2, W_ADDRESS2, W_DATA2, W_DATA_READY2, W_MAX_ENABLE2,
W_LOG_LIK2);
        VA3 : VITERBI_ALGO3
    PORT MAP(CLOCK, W_ENABLE3, W_ADDRESS3, W_DATA3, W_DATA_READY3, W_MAX_ENABLE3,
W_LOG_LIK3);
        VA4 : VITERBI_ALGO4
    PORT MAP(CLOCK, W_ENABLE4, W_ADDRESS4, W_DATA4, W_DATA_READY4, W_MAX_ENABLE4,
W_LOG_LIK4);
        OS0 : ROM_MODULE0
    PORT MAP(CLOCK, W_ENABLE0, W_ADDRESS0, W_DATA_READY0, W_DATA0);
    OS1 : ROM_MODULE1
    PORT MAP(CLOCK, W_ENABLE1, W_ADDRESS1, W_DATA_READY1, W_DATA1);
        OS2 : ROM_MODULE2
    PORT MAP(CLOCK, W_ENABLE2, W_ADDRESS2, W_DATA_READY2, W_DATA2);
        OS3 : ROM_MODULE3
    PORT MAP(CLOCK, W_ENABLE3, W_ADDRESS3, W_DATA_READY3, W_DATA3);
        OS4 : ROM_MODULE4
    PORT MAP(CLOCK, W_ENABLE4, W_ADDRESS4, W_DATA_READY4, W_DATA4);
    MAX_FIND : MAX_MODULE
    PORT MAP(CLOCK, W_MAX_ENABLE0, W_MAX_ENABLE1, W_MAX_ENABLE2, W_MAX_ENABLE3,
W_MAX_ENABLE4, W_LOG_LIK0, W_LOG_LIK1, W_LOG_LIK2, W_LOG_LIK3, W_LOG_LIK4,
RECOG_NO);
END VITERBI_SIM;

ARCHITECTURE ROM_MODULE0 OF ROM_MODULE0 IS

   TYPE ROM_ARRAY IS ARRAY(0 TO 31)       OF STD_LOGIC_VECTOR(7 DOWNTO 0);

   CONSTANT CONTENT: ROM_ARRAY := (
   0 => "00010001",
1 => "00010001",
2 => "00010010",
3 => "00010011",
4 => "00010011",
5 => "00010010",
6 => "00010010",
7 => "00010001",
8 => "00010111",
9 => "00011111",
10 => "00011011",
11 => "00000001",
12 => "00000001",
13 => "00000101",
14 => "00000011",
15 => "00001111",
16 => "00001101",
17 => "00000101",
18 => "00001101",
19 => "00001010",
```

```
20 => "00001110",
21 => "00001001",
22 => "00001001",
23 => "00001101",
24 => "00000101",
25 => "00001111",
26 => "00001101",
27 => "00001101",
28 => "00001111",
29 => "00000011",
30 => "00011111",
31 => "00011000"
      );

BEGIN
    PRO6 : PROCESS(CLOCK)
    BEGIN
        IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
              IF( ENABLE = '1') THEN
                 DATA_OUT <= CONTENT(CONV_INTEGER(ADDRESS));
                     DATA_READY <= '1';
               ELSE
             DATA_OUT <= "ZZZZZZZZ";
             DATA_READY <= '0';
                END IF;
           END IF;
    END PROCESS PRO6;
END ROM_MODULE0;

ARCHITECTURE ROM_MODULE1 OF ROM_MODULE1 IS

   TYPE ROM_ARRAY IS ARRAY(0 TO 31)       OF STD_LOGIC_VECTOR(7 DOWNTO 0);

   CONSTANT CONTENT: ROM_ARRAY := (
0 => "00000101",
1 => "00000101",
2 => "00000101",
3 => "00000101",
4 => "00000101",
5 => "00000101",
6 => "00000101",
7 => "00000101",
8 => "00000101",
9 => "00000001",
10 => "00010011",
11 => "00010001",
12 => "00010001",
13 => "00010101",
14 => "00010011",
15 => "00010101",
16 => "00010111",
17 => "00010101",
18 => "00010111",
19 => "00010111",
20 => "00010111",
21 => "00010111",
22 => "00010111",
```

```
23 => "00010111",
24 => "00010001",
25 => "00010101",
26 => "00010111",
27 => "00010111",
28 => "00010101",
29 => "00010100",
30 => "00001101",
31 => "00001000"



      );

BEGIN
    PRO5 : PROCESS(CLOCK)
    BEGIN
        IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
              IF( ENABLE = '1') THEN
                 DATA_OUT <= CONTENT(CONV_INTEGER(ADDRESS));
                     DATA_READY <= '1';
               ELSE
             DATA_OUT <= "ZZZZZZZZ";
             DATA_READY <= '0';
                END IF;
           END IF;
    END PROCESS PRO5;
END ROM_MODULE1;

ARCHITECTURE ROM_MODULE2 OF ROM_MODULE2 IS

   TYPE ROM_ARRAY IS ARRAY(0 TO 31)       OF STD_LOGIC_VECTOR(7 DOWNTO 0);

   CONSTANT CONTENT: ROM_ARRAY := (
0 => "00000011",
1 => "00000011",
2 => "00000011",
3 => "00000011",
4 => "00000011",
5 => "00000011",
6 => "00000011",
7 => "00000011",
8 => "00000001",
9 => "00000111",
10 => "00010101",
11 => "00011111",
12 => "00011010",
13 => "00011010",
14 => "00011111",
15 => "00011011",
16 => "00011100",
17 => "00011010",
18 => "00011100",
19 => "00011100",
20 => "00011100",
21 => "00011100",
22 => "00011100",
```

```
23 => "00011011",
24 => "00011010",
25 => "00011011",
26 => "00011100",
27 => "00011100",
28 => "00011100",
29 => "00100000",
30 => "00001101",
31 => "00001011"



      );

BEGIN
    PRO5 : PROCESS(CLOCK)
    BEGIN
        IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
              IF( ENABLE = '1') THEN
                 DATA_OUT <= CONTENT(CONV_INTEGER(ADDRESS));
                     DATA_READY <= '1';
               ELSE
             DATA_OUT <= "ZZZZZZZZ";
             DATA_READY <= '0';
                END IF;
           END IF;
    END PROCESS PRO5;
END ROM_MODULE2;

ARCHITECTURE ROM_MODULE3 OF ROM_MODULE3 IS

   TYPE ROM_ARRAY IS ARRAY(0 TO 31)       OF STD_LOGIC_VECTOR(7 DOWNTO 0);

   CONSTANT CONTENT: ROM_ARRAY := (
0 => "00001001",
1 => "00001001",
2 => "00001001",
3 => "00001001",
4 => "00001001",
5 => "00001001",
6 => "00001001",
7 => "00001001",
8 => "00001011",
9 => "00001111",
10 => "00010011",
11 => "00010010",
12 => "00010001",
13 => "00010001",
14 => "00010100",
15 => "00010001",
16 => "00010001",
17 => "00010001",
18 => "00010001",
19 => "00010001",
20 => "00010001",
21 => "00010001",
22 => "00010001",
```

```
23 => "00010001",
24 => "00010001",
25 => "00010001",
26 => "00010001",
27 => "00010001",
28 => "00010001",
29 => "00010100",
30 => "00011100",
31 => "00001100"



      );

BEGIN
    PRO5 : PROCESS(CLOCK)
    BEGIN
        IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
              IF( ENABLE = '1') THEN
                 DATA_OUT <= CONTENT(CONV_INTEGER(ADDRESS));
                     DATA_READY <= '1';
               ELSE
             DATA_OUT <= "ZZZZZZZZ";
             DATA_READY <= '0';
                END IF;
           END IF;
    END PROCESS PRO5;
END ROM_MODULE3;

ARCHITECTURE ROM_MODULE4 OF ROM_MODULE4 IS

   TYPE ROM_ARRAY IS ARRAY(0 TO 31)       OF STD_LOGIC_VECTOR(7 DOWNTO 0);

   CONSTANT CONTENT: ROM_ARRAY := (
0 => "00010001",
1 => "00010001",
2 => "00010001",
3 => "00010010",
4 => "00010010",
5 => "00010001",
6 => "00010001",
7 => "00010001",
8 => "00010010",
9 => "00011000",
10 => "00001100",
11 => "00001010",
12 => "00001010",
13 => "00000110",
14 => "00001110",
15 => "00000001",
16 => "00000011",
17 => "00000110",
18 => "00000011",
19 => "00000011",
20 => "00000011",
21 => "00000011",
22 => "00000011",
```

```
23 => "00000011",
24 => "00000101",
25 => "00000001",
26 => "00000011",
27 => "00000011",
28 => "00000010",
29 => "00010000",
30 => "00011110",
31 => "00011110"



      );

BEGIN
    PRO5 : PROCESS(CLOCK)
    BEGIN
        IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
              IF( ENABLE = '1') THEN
                 DATA_OUT <= CONTENT(CONV_INTEGER(ADDRESS));
                     DATA_READY <= '1';
               ELSE
             DATA_OUT <= "ZZZZZZZZ";
             DATA_READY <= '0';
                END IF;
           END IF;
    END PROCESS PRO5;
END ROM_MODULE4;

ARCHITECTURE VITERBI_ALGO0 OF VITERBI_ALGO0 IS

   TYPE DATAB IS ARRAY(1 TO 192) OF INTEGER;
       TYPE SANDY IS ARRAY(1 TO 6) OF INTEGER;
       CONSTANT B : DATAB := (
                     1 =>  -6386,
          2 => -9999999,
          3 => -317622,
          4 => -9999999,
          5 => -720978,
          6 => -9999999,
          7 => -9999999,
          8 => -9999999,
          9 => -9999999,
          10 => -9999999,
          11 => -9999999,
          12 => -9999999,
          13 => -9999999,
          14 => -9999999,
          15 => -9999999,
          16 => -9999999,
          17 =>  -3560,
          18 =>  -2166,
          19 =>  -3020,
          20 =>  -2142,
          21 =>  -1891,
          22 =>  -4625,
          23 =>  -1448,
```

```
24 =>  -1341,
25 => -139393,
26 =>  -5289,
27 =>  -5786,
28 =>  -6231,
29 =>  -4039,
30 =>  -5520,
31 =>  -9926,
32 => -317871,
33 =>  -8210,
34 => -9999999,
35 => -198049,
36 => -259317,
37 => -341863,
38 => -621536,
39 => -510972,
40 => -127824,
41 => -9999999,
42 => -9999999,
43 => -9999999,
44 => -9999999,
45 => -9999999,
46 => -9999999,
47 => -9999999,
48 => -9999999,
49 =>  -2249,
50 =>  -5813,
51 =>  -1402,
52 =>  -6149,
53 => -10341,
54 => -72747,
55 =>  -1713,
56 =>  -1406,
57 => -27097,
58 =>  -2393,
59 =>  -4953,
60 => -26309,
61 =>  -2320,
62 =>  -6291,
63 =>  -3979,
64 => -35956,
65 =>  -5538,
66 => -85424,
67 => -20549,
68 => -48218,
69 => -44934,
70 => -70981,
71 => -77316,
72 => -43503,
73 => -9999999,
74 => -475310,
75 => -9999999,
76 => -9999999,
77 => -9999999,
78 => -622503,
79 => -514111,
80 => -625988,
```

```
81 => -578843,
82 => -460003,
83 => -51243,
84 => -724225,
85 => -57644,
86 => -113530,
87 => -30560,
88 => -11886,
89 =>   -714,
90 =>  -3769,
91 =>  -1558,
92 =>  -2929,
93 =>  -1754,
94 => -53594,
95 =>  -3150,
96 =>  -5712,
97 =>  -1838,
98 =>  -5579,
99 =>  -2649,
100 =>  -3193,
101 =>  -2146,
102 =>  -2801,
103 =>  -3133,
104 =>  -3640,
105 =>  -3297,
106 =>  -3633,
107 =>  -3494,
108 =>  -4550,
109 =>  -2801,
110 =>  -3990,
111 =>  -3297,
112 =>  -4231,
113 => -9999999,
114 => -9999999,
115 => -9999999,
116 => -9999999,
117 => -693768,
118 => -122837,
119 => -245229,
120 => -29936,
121 => -27300,
122 => -15347,
123 =>  -4571,
124 =>  -2153,
125 =>  -4199,
126 =>  -6528,
127 =>  -2845,
128 =>  -3184,
129 =>  -4741,
130 =>  -1852,
131 =>  -7051,
132 =>  -4336,
133 =>  -4263,
134 =>  -9473,
135 => -16392,
136 =>  -6144,
137 => -73601,
```

```
   138 => -194392,
   139 => -227078,
   140 => -502910,
   141 => -36187,
   142 => -466098,
   143 => -27652,
   144 => -50370,
   145 => -9999999,
   146 => -9999999,
   147 => -9999999,
   148 => -54126,
   149 => -45696,
   150 => -59517,
   151 => -104484,
   152 =>  -4848,
   153 =>  -6273,
   154 =>  -1421,
   155 =>  -1664,
   156 =>  -3358,
   157 =>  -3267,
   158 =>  -3443,
   159 =>  -1564,
   160 =>  -3007,
   161 => -52294,
   162 => -65957,
   163 => -96188,
   164 => -30746,
   165 => -67915,
   166 => -282110,
   167 => -126358,
   168 => -47179,
   169 => -9999999,
   170 => -9999999,
   171 => -9999999,
   172 => -9999999,
   173 => -79574,
   174 => -9999999,
   175 => -94759,
   176 => -465763,
   177 => -9999999,
   178 =>  -4915,
   179 => -9999999,
   180 =>  -4222,
   181 =>  -4915,
   182 =>  -1657,
   183 =>  -3123,
   184 =>  -1410,
   185 => -32908,
   186 =>  -3511,
   187 => -50136,
   188 =>  -5944,
   189 =>  -3242,
   190 =>   -977,
   191 =>  -3182,
   192 =>  -6137
               );
FUNCTION FIND_MAX1( F : SANDY) RETURN INTEGER IS
```

```
     VARIABLE P : INTEGER := 0;
     BEGIN
     P := F(1);
     FOR R IN 2 TO 6 LOOP
     IF (P < F(R)) THEN P := F(R); END IF;
END LOOP;
     RETURN P;
     END FIND_MAX1;
     FUNCTION FIND_MAX2(X,Y:INTEGER) RETURN INTEGER IS
     BEGIN
     IF(X>Y) THEN RETURN X; ELSE RETURN Y;END IF;
     END FIND_MAX2;
     BEGIN
PRO1 : PROCESS(CLOCK)
   VARIABLE I : INTEGER := 0;
   BEGIN
       IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
     IF(I < 35) THEN
         ENABLE <= '1';
         ADDRESS <= CONV_STD_LOGIC_VECTOR(I, 5);
             I := I+1;
             ELSE
             ENABLE <= '0';
             ADDRESS <= "ZZZZZ";
             END IF;
          END IF;
END PROCESS PRO1;
PRO2 : PROCESS(CLOCK)
   VARIABLE D  : SANDY;
   VARIABLE O  : INTEGER RANGE 1 TO 32;
   VARIABLE K : INTEGER := 1;

          BEGIN
   IF(CLOCK'EVENT AND CLOCK  = '1') THEN
      IF(DATA_READY = '1') THEN
        IF(K < 33) THEN
          O := CONV_INTEGER(DATA_IN);
          IF(K = 1) THEN
            -- INITIALIZATION
                    D(1) := B(O);
                    D(2) := B(32 + O) - 9999999;
                    D(3) := B(64 + O) - 9999999;
                    D(4) := B(96 + O) - 9999999;
                    D(5) := B(128 + O) - 9999999;
                    D(6) := B(160 + O) - 9999999;
                    K := K + 1;
                ELSE
                D(6) := FIND_MAX2((D(5)-2050),D(6)) + B(160+O);
                                D(5) := FIND_MAX2((D(4)-2082),(D(5)-
138)) + B(128+O);
                                D(4) := FIND_MAX2((D(3)-1564),(D(4)-
133)) + B(96+O);
                                D(3) := FIND_MAX2((D(2)-1641),(D(3)-
235)) + B(64+O);
                                D(2) := FIND_MAX2((D(1)-1809),(D(2)-
215)) + B(32+O);
                                D(1) := D(1) - 179 + B(O);
```

```
                                        K := K + 1;
                    END IF;
                    ELSIF(K = 33) THEN
--            -- TERMINATION
                    LOG_LIK <= CONV_STD_LOGIC_VECTOR(FIND_MAX1(D),32);
                     MAX_ENABLE <= '1';
                    END IF;
                    END IF;
             END IF;
      END PROCESS PRO2;
  END VITERBI_ALGO0;

  ARCHITECTURE VITERBI_ALGO1 OF VITERBI_ALGO1 IS

  TYPE DATAB IS ARRAY(1 TO 192) OF INTEGER;
      TYPE SANDY IS ARRAY(1 TO 6) OF INTEGER;
      CONSTANT B : DATAB := (
                     1 =>  -3336,
        2 =>  -4320,
        3 =>  -5871,
        4 =>  -4318,
        5 =>  -3119,
        6 =>  -1492,
        7 =>  -2066,
        8 =>  -1695,
        9 =>  -7200,
        10 => -16484,
        11 =>  -2355,
        12 =>  -1689,
        13 =>  -5602,
        14 =>  -3278,
        15 => -13831,
        16 =>  -4105,
        17 => -14377,
        18 => -24119,
        19 =>  -4646,
        20 => -17483,
        21 => -109089,
        22 => -126645,
        23 => -147692,
        24 => -63404,
        25 => -11951,
        26 => -38996,
        27 =>  -4855,
        28 =>  -9496,
        29 => -26688,
        30 => -19618,
        31 => -36284,
        32 => -53069,
        33 =>  -7247,
        34 =>  -8985,
        35 =>  -5103,
        36 =>  -9163,
        37 => -45592,
        38 => -24033,
        39 =>  -6453,
        40 =>  -2893,
```

```
41 =>  -2996,
42 =>  -2042,
43 =>  -2965,
44 =>  -2308,
45 =>  -3208,
46 =>  -5517,
47 =>  -2192,
48 =>  -1825,
49 =>  -5477,
50 =>  -6927,
51 => -13583,
52 =>  -5100,
53 => -20623,
54 =>  -4997,
55 =>  -6157,
56 =>  -4849,
57 =>  -3818,
58 =>  -4271,
59 =>  -7353,
60 =>  -2488,
61 =>  -3136,
62 =>  -3107,
63 =>  -3940,
64 =>  -3424,
65 =>  -5971,
66 => -21791,
67 =>  -5064,
68 => -12737,
69 => -141604,
70 => -32818,
71 =>  -6744,
72 => -11286,
73 =>  -1945,
74 =>  -6050,
75 =>  -3404,
76 =>  -6657,
77 =>  -4841,
78 =>  -5976,
79 => -12897,
80 =>  -5903,
81 =>  -1946,
82 =>  -2497,
83 =>  -2256,
84 =>  -2178,
85 =>  -2726,
86 =>  -4057,
87 =>  -3878,
88 =>  -5332,
89 =>  -2948,
90 =>  -2481,
91 =>  -3456,
92 =>  -3782,
93 =>  -5023,
94 =>  -3757,
95 =>  -3689,
96 => -14171,
97 =>  -3671,
```

```
98 =>  -4327,
99 =>  -1510,
100 =>  -4035,
101 => -50091,
102 => -28098,
103 =>  -3914,
104 =>  -4912,
105 =>  -2755,
106 => -26243,
107 =>  -4515,
108 =>  -4089,
109 =>  -2273,
110 =>  -5781,
111 =>  -4175,
112 =>  -3093,
113 =>  -7394,
114 =>  -5206,
115 =>  -2572,
116 =>  -3409,
117 => -10887,
118 =>  -8466,
119 => -36014,
120 => -43861,
121 =>  -1915,
122 =>  -3741,
123 =>  -8161,
124 =>  -2231,
125 => -21565,
126 =>  -3712,
127 =>  -7547,
128 =>  -4034,
129 =>  -2168,
130 =>  -1294,
131 =>  -3422,
132 =>  -1607,
133 => -11104,
134 =>  -5869,
135 =>  -4825,
136 =>  -2528,
137 =>  -6104,
138 =>  -6235,
139 => -57192,
140 => -14168,
141 =>  -2120,
142 =>  -3902,
143 => -16002,
144 =>  -2865,
145 =>  -5627,
146 => -11425,
147 =>  -5394,
148 =>  -4926,
149 => -34112,
150 => -28285,
151 => -123865,
152 => -287083,
153 =>  -2963,
154 =>  -4714,
```

```
        155 =>  -6029,
        156 =>  -5182,
        157 => -105850,
        158 =>  -6646,
        159 => -31621,
        160 =>  -7293,
        161 =>  -5364,
        162 =>  -9237,
        163 => -29318,
        164 => -19441,
        165 =>  -2337,
        166 =>  -2018,
        167 =>  -3359,
        168 =>  -1157,
        169 => -62630,
        170 => -14528,
        171 => -354289,
        172 =>  -2769,
        173 =>  -6105,
        174 =>  -2401,
        175 =>  -4627,
        176 =>  -2305,
        177 => -35726,
        178 => -63354,
        179 => -57533,
        180 =>  -6557,
        181 => -120485,
        182 => -98751,
        183 => -243307,
        184 => -9999999,
        185 => -12197,
        186 =>  -4046,
        187 =>  -2229,
        188 =>  -4846,
        189 =>  -5600,
        190 =>  -4334,
        191 => -77537,
        192 => -26304
                    );
    FUNCTION FIND_MAX1( F : SANDY) RETURN INTEGER IS
    VARIABLE P : INTEGER := 0;
    BEGIN
    P := F(1);
    FOR R IN 2 TO 6 LOOP
    IF (P < F(R)) THEN P := F(R); END IF;
END LOOP;
    RETURN P;
    END FIND_MAX1;
    FUNCTION FIND_MAX2(X,Y:INTEGER) RETURN INTEGER IS
    BEGIN
    IF(X>Y) THEN RETURN X; ELSE RETURN Y;END IF;
    END FIND_MAX2;
    BEGIN
PRO3 : PROCESS(CLOCK)
   VARIABLE I : INTEGER := 0;
   BEGIN
       IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
```

```
        IF(I < 35) THEN
            ENABLE <= '1';
            ADDRESS <= CONV_STD_LOGIC_VECTOR(I, 5);
                I := I+1;
                ELSE
                ENABLE <= '0';
                ADDRESS <= "ZZZZZ";
                END IF;
             END IF;
   END PROCESS PRO3;
   PRO4 : PROCESS(CLOCK)
      VARIABLE D  : SANDY;
      VARIABLE O  : INTEGER RANGE 1 TO 32;
      VARIABLE K : INTEGER := 1;

              BEGIN

      IF(CLOCK'EVENT AND CLOCK = '1') THEN
          IF(DATA_READY = '1') THEN
             IF(K < 33) THEN
                O := CONV_INTEGER(DATA_IN);
                IF(K = 1) THEN
                 -- INITIALIZATION
                           D(1) := B(O);
                           D(2) := B(32 + O) - 9999999;
                           D(3) := B(64 + O) - 9999999;
                           D(4) := B(96 + O) - 9999999;
                           D(5) := B(128 + O) - 9999999;
                           D(6) := B(160 + O) - 9999999;
                           K := K + 1;
                    ELSE
                    D(6) := FIND_MAX2((D(5)-1754),D(6)) + B(160+O);
                                        D(5) := FIND_MAX2((D(4)-1339),(D(5)-
190)) + B(128+O);
                                        D(4) := FIND_MAX2((D(3)-1810),(D(4)-
304)) + B(96+O);
                                        D(3) := FIND_MAX2((D(2)-1863),(D(3)-
179)) + B(64+O);
                                        D(2) := FIND_MAX2((D(1)-1825),(D(2)-
169)) + B(32+O);
                                        D(1) := D(1) - 176 + B(O);
                                        K := K + 1;
                    END IF;
                    ELSIF(K = 33) THEN
--            -- TERMINATION
                    LOG_LIK <= CONV_STD_LOGIC_VECTOR(FIND_MAX1(D),32);
                     MAX_ENABLE <= '1';
                    END IF;
                    END IF;
             END IF;
       END PROCESS PRO4;
    END VITERBI_ALGO1;

        ARCHITECTURE VITERBI_ALGO2 OF VITERBI_ALGO2 IS

   TYPE DATAB IS ARRAY(1 TO 192) OF INTEGER;
       TYPE SANDY IS ARRAY(1 TO 6) OF INTEGER;
```

```
      CONSTANT B : DATAB := (
                    1 =>  -3375,
2 =>  -2814,
3 =>  -1670,
4 =>  -1261,
5 =>  -3046,
6 =>  -1749,
7 =>  -3707,
8 => -10914,
9 => -173998,
10 => -201716,
11 =>  -4423,
12 =>  -4983,
13 =>  -4067,
14 =>  -4290,
15 =>  -6369,
16 => -9999999,
17 => -28077,
18 => -23058,
19 =>  -2735,
20 =>  -4715,
21 =>  -3665,
22 =>  -5340,
23 =>  -6369,
24 =>  -4288,
25 => -15388,
26 =>  -5854,
27 => -11542,
28 => -487873,
29 => -11293,
30 =>  -4996,
31 =>  -4923,
32 => -145733,
33 => -14909,
34 =>  -9894,
35 =>  -8830,
36 =>  -5115,
37 =>  -2444,
38 =>  -4152,
39 => -11029,
40 =>  -5539,
41 => -201932,
42 => -184643,
43 => -50579,
44 => -20052,
45 => -27717,
46 => -32159,
47 => -153115,
48 => -460098,
49 =>  -2001,
50 =>  -1989,
51 =>  -3267,
52 =>  -1960,
53 =>  -3539,
54 =>  -2946,
55 => -30471,
56 => -14982,
```

```
57 =>  -2841,
58 =>  -5426,
59 =>  -3842,
60 => -187566,
61 =>  -2310,
62 =>  -1789,
63 =>  -5484,
64 => -36336,
65 =>  -4733,
66 =>  -4775,
67 =>  -2290,
68 =>  -2166,
69 =>  -3671,
70 =>  -1676,
71 =>  -3126,
72 =>  -2027,
73 => -175040,
74 => -71921,
75 => -16161,
76 =>  -5076,
77 => -11767,
78 => -43050,
79 => -239349,
80 => -155183,
81 => -13163,
82 =>  -6882,
83 =>  -2074,
84 =>  -3427,
85 =>  -2913,
86 =>  -2260,
87 =>  -6230,
88 =>  -5745,
89 => -19641,
90 =>  -5188,
91 => -11058,
92 => -51145,
93 =>  -4694,
94 =>  -3502,
95 =>  -6202,
96 =>  -5970,
97 =>  -5479,
98 =>  -1801,
99 =>  -4493,
100 =>  -5606,
101 =>  -4548,
102 => -11692,
103 =>  -1047,
104 =>  -4459,
105 => -34923,
106 => -25799,
107 =>  -4956,
108 =>  -3390,
109 =>  -3521,
110 =>  -4685,
111 => -44603,
112 => -34989,
113 => -22641,
```

```
114 =>  -3761,
115 =>  -5374,
116 =>  -5085,
117 =>  -2442,
118 =>  -5078,
119 =>  -7859,
120 =>  -1845,
121 => -86163,
122 =>  -3711,
123 =>  -6067,
124 =>  -5640,
125 =>  -8601,
126 => -10226,
127 =>  -3254,
128 =>  -4814,
129 =>  -9658,
130 =>  -3742,
131 =>  -5874,
132 =>  -5083,
133 => -24043,
134 => -66423,
135 =>  -3123,
136 =>  -4510,
137 =>  -5959,
138 =>  -4700,
139 =>  -2484,
140 =>  -1414,
141 =>  -1241,
142 =>  -2051,
143 =>  -4517,
144 =>  -6324,
145 => -121667,
146 =>  -5987,
147 => -51013,
148 => -81266,
149 => -31898,
150 => -11094,
151 =>  -2786,
152 =>  -3231,
153 => -365256,
154 => -11014,
155 => -18301,
156 =>  -4459,
157 => -81324,
158 =>  -6150,
159 => -13429,
160 =>  -3648,
161 =>  -2966,
162 =>  -4452,
163 =>  -9879,
164 => -19371,
165 => -61629,
166 => -57294,
167 => -23680,
168 => -29064,
169 =>  -1385,
170 =>  -1716,
```

```
171 =>  -1524,
172 =>  -4051,
173 =>  -3510,
174 =>  -4212,
175 =>  -2380,
176 =>  -2008,
177 => -392061,
178 => -82836,
179 => -73066,
180 => -479528,
181 => -155195,
182 => -120730,
183 => -32442,
184 => -19395,
185 => -9999999,
186 => -262223,
187 => -235039,
188 => -21746,
189 => -471866,
190 => -15127,
191 => -85912,
192 => -15059


 );
      FUNCTION FIND_MAX1( F : SANDY) RETURN INTEGER IS
      VARIABLE P : INTEGER := 0;
      BEGIN
      P := F(1);
      FOR R IN 2 TO 6 LOOP
      IF (P < F(R)) THEN P := F(R); END IF;
  END LOOP;
      RETURN P;
      END FIND_MAX1;
      FUNCTION FIND_MAX2(X,Y:INTEGER) RETURN INTEGER IS
      BEGIN
      IF(X>Y) THEN RETURN X; ELSE RETURN Y;END IF;
      END FIND_MAX2;
      BEGIN
  PRO3 : PROCESS(CLOCK)
     VARIABLE I : INTEGER := 0;
     BEGIN
         IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
       IF(I < 35) THEN
           ENABLE <= '1';
           ADDRESS <= CONV_STD_LOGIC_VECTOR(I, 5);
               I := I+1;
               ELSE
               ENABLE <= '0';
               ADDRESS <= "ZZZZZ";
               END IF;
            END IF;
  END PROCESS PRO3;
  PRO4 : PROCESS(CLOCK)
     VARIABLE D  : SANDY;
     VARIABLE O  : INTEGER RANGE 1 TO 32;
     VARIABLE K : INTEGER := 1;
```

```
            BEGIN

     IF(CLOCK'EVENT AND CLOCK = '1') THEN
          IF(DATA_READY = '1') THEN
             IF(K < 33) THEN
                O := CONV_INTEGER(DATA_IN);
                IF(K = 1) THEN
                 -- INITIALIZATION
                           D(1) := B(O);
                           D(2) := B(32 + O) - 9999999;
                           D(3) := B(64 + O) - 9999999;
                           D(4) := B(96 + O) - 9999999;
                           D(5) := B(128 + O) - 9999999;
                           D(6) := B(160 + O) - 9999999;
                           K := K + 1;
                     ELSE
                     D(6) := FIND_MAX2((D(5)-1856),D(6)) + B(160+O);
                                      D(5) := FIND_MAX2((D(4)-1573),(D(5)-
170)) + B(128+O);
                                      D(4) := FIND_MAX2((D(3)-1583),(D(4)-
232)) + B(96+O);
                                      D(3) := FIND_MAX2((D(2)-1902),(D(3)-
230)) + B(64+O);
                                      D(2) := FIND_MAX2((D(1)-1946),(D(2)-
162)) + B(32+O);
                                      D(1) := D(1) - 154 + B(O);
                                      K := K + 1;
                     END IF;
                     ELSIF(K = 33) THEN
--            -- TERMINATION
                     LOG_LIK <= CONV_STD_LOGIC_VECTOR(FIND_MAX1(D),32);
                      MAX_ENABLE <= '1';
                     END IF;
                     END IF;
               END IF;
       END PROCESS PRO4;
    END VITERBI_ALGO2;

         ARCHITECTURE VITERBI_ALGO3 OF VITERBI_ALGO3 IS

   TYPE DATAB IS ARRAY(1 TO 192) OF INTEGER;
       TYPE SANDY IS ARRAY(1 TO 6) OF INTEGER;
       CONSTANT B : DATAB := (
                        1 => -737148,
2 => -137978,
3 => -16438,
4 => -24846,
5 => -430770,
6 => -161195,
7 => -19289,
8 => -9999999,
9 =>  -1605,
10 =>   -932,
11 =>  -2714,
12 =>  -5081,
13 =>  -5081,
```

```
14 =>  -6282,
15 =>  -1891,
16 =>  -2778,
17 =>  -4567,
18 => -16815,
19 => -21199,
20 =>  -5763,
21 => -27846,
22 => -22864,
23 =>  -4088,
24 =>  -4052,
25 =>  -5486,
26 =>  -5081,
27 =>  -4793,
28 =>  -4793,
29 =>  -3698,
30 =>  -5486,
31 =>  -5486,
32 =>  -5486,
33 =>  -2823,
34 =>  -2766,
35 =>  -4683,
36 =>  -4356,
37 =>  -3054,
38 =>  -2794,
39 =>  -2795,
40 => -13180,
41 => -37198,
42 => -28972,
43 => -44502,
44 => -243139,
45 => -34743,
46 =>  -8552,
47 =>  -4617,
48 => -14052,
49 =>  -4560,
50 =>  -2712,
51 =>  -1973,
52 =>  -2271,
53 =>  -1967,
54 =>  -2176,
55 =>  -2624,
56 =>  -3472,
57 => -461064,
58 => -223892,
59 => -28080,
60 => -307660,
61 =>  -9454,
62 => -15377,
63 => -437331,
64 => -20605,
65 => -90815,
66 => -15292,
67 =>  -1606,
68 =>  -1951,
69 => -35159,
70 => -15784,
```

```
71 =>  -4593,
72 =>  -3375,
73 => -149150,
74 =>  -8661,
75 => -161560,
76 => -294846,
77 =>  -1547,
78 =>  -1387,
79 =>  -3605,
80 => -23404,
81 => -149730,
82 => -29553,
83 =>  -5156,
84 =>  -3749,
85 => -13426,
86 =>  -3218,
87 =>  -3386,
88 =>  -3938,
89 => -390787,
90 => -117976,
91 => -153735,
92 => -514693,
93 => -22746,
94 => -39061,
95 => -159704,
96 => -162038,
97 => -9999999,
98 => -181816,
99 => -142172,
100 => -104761,
101 => -586372,
102 => -249099,
103 => -14747,
104 => -29590,
105 =>  -6324,
106 =>  -4137,
107 => -40511,
108 =>  -6324,
109 =>  -3543,
110 =>  -3036,
111 =>  -1457,
112 =>  -1840,
113 => -658991,
114 => -74654,
115 =>  -6804,
116 =>  -5905,
117 => -73656,
118 =>  -4683,
119 =>  -1079,
120 =>  -2927,
121 => -26541,
122 =>  -5698,
123 => -45585,
124 => -112576,
125 =>  -2328,
126 =>  -5424,
127 => -47270,
```

```
128 => -26336,
129 => -9999999,
130 => -9999999,
131 => -9999999,
132 => -9999999,
133 => -9999999,
134 => -9999999,
135 => -97172,
136 => -266746,
137 => -176009,
138 => -94824,
139 =>  -5651,
140 => -20562,
141 => -63951,
142 => -36361,
143 => -21025,
144 =>  -3563,
145 => -9999999,
146 => -241636,
147 => -28591,
148 =>  -2135,
149 => -611193,
150 => -28276,
151 => -10060,
152 =>  -1097,
153 =>  -4054,
154 =>  -4730,
155 =>  -4498,
156 => -31143,
157 =>  -2659,
158 =>  -1121,
159 =>  -2879,
160 =>  -3636,
161 => -9999999,
162 => -9999999,
163 => -9999999,
164 => -9999999,
165 => -9999999,
166 => -9999999,
167 => -9999999,
168 => -9999999,
169 =>  -6586,
170 =>  -5892,
171 =>  -3897,
172 =>  -3450,
173 => -698496,
174 => -375058,
175 => -70492,
176 => -36300,
177 => -9999999,
178 => -9999999,
179 => -363667,
180 => -16950,
181 => -9999999,
182 => -380281,
183 => -97303,
184 =>  -6413,
```

```
185 =>  -1974,
186 =>  -1630,
187 =>  -1734,
188 =>  -1710,
189 => -15374,
190 =>  -4589,
191 =>  -2467,
192 =>  -1864
                          );
      FUNCTION FIND_MAX1( F : SANDY) RETURN INTEGER IS
      VARIABLE P : INTEGER := 0;
      BEGIN
      P := F(1);
      FOR R IN 2 TO 6 LOOP
      IF (P < F(R)) THEN P := F(R); END IF;
  END LOOP;
      RETURN P;
      END FIND_MAX1;
      FUNCTION FIND_MAX2(X,Y:INTEGER) RETURN INTEGER IS
      BEGIN
      IF(X>Y) THEN RETURN X; ELSE RETURN Y;END IF;
      END FIND_MAX2;
      BEGIN
  PRO3 : PROCESS(CLOCK)
     VARIABLE I : INTEGER := 0;
     BEGIN
         IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
        IF(I < 35) THEN
            ENABLE <= '1';
            ADDRESS <= CONV_STD_LOGIC_VECTOR(I, 5);
                I := I+1;
                ELSE
                ENABLE <= '0';
                ADDRESS <= "ZZZZZ";
                END IF;
             END IF;
  END PROCESS PRO3;
  PRO4 : PROCESS(CLOCK)
     VARIABLE D  : SANDY;
     VARIABLE O  : INTEGER RANGE 1 TO 32;
     VARIABLE K : INTEGER := 1;

             BEGIN

     IF(CLOCK'EVENT AND CLOCK = '1') THEN
         IF(DATA_READY = '1') THEN
            IF(K < 33) THEN
               O := CONV_INTEGER(DATA_IN);
               IF(K = 1) THEN
                -- INITIALIZATION
                          D(1) := B(O);
                          D(2) := B(32 + O) - 9999999;
                          D(3) := B(64 + O) - 9999999;
                          D(4) := B(96 + O) - 9999999;
                          D(5) := B(128 + O) - 9999999;
                          D(6) := B(160 + O) - 9999999;
                          K := K + 1;
```

```
                    ELSE
                    D(6) := FIND_MAX2((D(5)-1315),D(6)) + B(160+O);
                                        D(5) := FIND_MAX2((D(4)-1719),(D(5)-
313)) + B(128+O);
                                        D(4) := FIND_MAX2((D(3)-1603),(D(4)-
198)) + B(96+O);
                                        D(3) := FIND_MAX2((D(2)-1744),(D(3)-
225)) + B(64+O);
                                        D(2) := FIND_MAX2((D(1)-1763),(D(2)-
192)) + B(32+O);
                                        D(1) := D(1) - 188 + B(O);
                                        K := K + 1;
                    END IF;
                    ELSIF(K = 33) THEN
--            -- TERMINATION
                    LOG_LIK <= CONV_STD_LOGIC_VECTOR(FIND_MAX1(D),32);
                     MAX_ENABLE <= '1';
                    END IF;
                    END IF;
             END IF;
      END PROCESS PRO4;
   END VITERBI_ALGO3;

       ARCHITECTURE VITERBI_ALGO4 OF VITERBI_ALGO4 IS

  TYPE DATAB IS ARRAY(1 TO 192) OF INTEGER;
      TYPE SANDY IS ARRAY(1 TO 6) OF INTEGER;
      CONSTANT B : DATAB := (
                     1 => -210560,
2 => -229949,
3 => -454028,
4 => -119061,
5 => -41878,
6 => -57779,
7 => -343500,
8 => -187670,
9 => -25112,
10 => -12983,
11 => -14965,
12 =>  -4968,
13 => -24197,
14 => -13970,
15 => -214052,
16 =>  -6342,
17 =>  -1408,
18 =>  -1226,
19 =>  -1480,
20 =>  -2834,
21 => -19640,
22 =>  -4275,
23 =>  -6465,
24 =>  -2431,
25 =>  -5218,
26 =>  -4263,
27 =>  -6342,
28 =>  -4956,
29 =>  -4956,
```

```
30 =>  -4397,
31 =>  -4263,
32 =>  -6342,
33 => -24788,
34 => -20603,
35 => -46030,
36 => -51882,
37 =>  -5245,
38 => -14146,
39 => -23818,
40 => -25003,
41 =>  -1721,
42 =>  -4183,
43 =>  -1824,
44 =>  -3715,
45 =>  -6240,
46 =>  -5674,
47 => -15982,
48 => -37193,
49 => -19509,
50 =>  -7448,
51 =>  -6763,
52 =>  -4915,
53 =>  -1365,
54 =>  -2561,
55 =>  -1444,
56 =>  -3451,
57 => -12285,
58 => -13568,
59 => -59503,
60 => -93174,
61 => -94736,
62 => -53774,
63 => -125525,
64 => -105128,
65 =>  -2900,
66 =>  -3725,
67 =>  -3782,
68 =>  -4682,
69 =>  -2655,
70 =>  -2521,
71 =>  -3694,
72 =>  -3844,
73 =>  -4628,
74 =>  -2146,
75 => -17446,
76 =>  -1688,
77 =>  -2555,
78 =>  -3245,
79 =>  -4559,
80 =>  -5376,
81 => -62329,
82 => -37187,
83 =>  -9994,
84 =>  -7132,
85 => -13214,
86 =>  -1673,
```

```
87 =>  -8086,
88 =>  -3385,
89 =>  -4071,
90 =>  -6273,
91 =>  -5290,
92 => -10721,
93 => -18408,
94 =>  -6291,
95 => -63152,
96 => -17382,
97 => -16051,
98 =>  -7375,
99 => -11757,
100 =>  -6050,
101 => -25303,
102 =>  -5184,
103 =>  -5526,
104 =>  -4609,
105 =>  -6128,
106 =>  -6388,
107 => -64775,
108 =>  -2907,
109 =>  -2516,
110 =>  -2047,
111 =>  -3964,
112 =>  -2730,
113 => -199292,
114 =>  -6162,
115 =>  -2985,
116 =>  -2656,
117 =>  -6165,
118 =>  -3410,
119 =>  -4312,
120 =>  -1782,
121 =>  -1729,
122 =>  -3157,
123 =>  -3470,
124 =>  -4476,
125 =>  -4115,
126 =>  -5821,
127 => -20137,
128 =>  -5733,
129 => -59269,
130 => -40339,
131 => -68542,
132 => -22066,
133 => -116810,
134 => -54614,
135 => -12980,
136 => -19388,
137 => -57744,
138 => -38926,
139 => -203024,
140 => -16522,
141 =>  -5302,
142 =>  -3946,
143 =>  -5013,
```

```
144 =>  -3775,
145 => -479106,
146 => -15233,
147 => -32267,
148 =>  -4233,
149 => -11601,
150 => -17928,
151 =>  -5327,
152 =>  -5417,
153 =>  -2967,
154 =>  -1049,
155 =>  -2150,
156 =>  -2345,
157 =>  -1451,
158 =>  -4544,
159 =>  -3112,
160 =>  -3975,
161 => -329318,
162 => -214037,
163 => -283768,
164 => -218701,
165 => -460366,
166 => -244560,
167 => -37124,
168 => -124863,
169 => -234887,
170 => -133941,
171 => -587552,
172 => -56123,
173 => -24059,
174 =>  -5901,
175 =>  -7777,
176 =>  -8398,
177 => -9999999,
178 =>  -5917,
179 => -142702,
180 => -11563,
181 => -34462,
182 => -110871,
183 => -25971,
184 =>  -6610,
185 =>  -6569,
186 =>  -5840,
187 =>  -2952,
188 =>  -2020,
189 =>  -2317,
190 =>  -1378,
191 =>  -1265,
192 =>  -1770


                );
    FUNCTION FIND_MAX1( F : SANDY) RETURN INTEGER IS
    VARIABLE P : INTEGER := 0;
    BEGIN
    P := F(1);
    FOR R IN 2 TO 6 LOOP
```

```
        IF (P < F(R)) THEN P := F(R); END IF;
    END LOOP;
        RETURN P;
        END FIND_MAX1;
        FUNCTION FIND_MAX2(X,Y:INTEGER) RETURN INTEGER IS
        BEGIN
        IF(X>Y) THEN RETURN X; ELSE RETURN Y;END IF;
        END FIND_MAX2;
        BEGIN
    PRO3 : PROCESS(CLOCK)
       VARIABLE I : INTEGER := 0;
       BEGIN
           IF( CLOCK'EVENT AND CLOCK = '1' ) THEN
          IF(I < 35) THEN
              ENABLE <= '1';
              ADDRESS <= CONV_STD_LOGIC_VECTOR(I, 5);
                  I := I+1;
                  ELSE
                  ENABLE <= '0';
                  ADDRESS <= "ZZZZZ";
                  END IF;
               END IF;
    END PROCESS PRO3;
    PRO4 : PROCESS(CLOCK)
       VARIABLE D  : SANDY;
       VARIABLE O  : INTEGER RANGE 1 TO 32;
       VARIABLE K : INTEGER := 1;

                BEGIN

       IF(CLOCK'EVENT AND CLOCK = '1') THEN
            IF(DATA_READY = '1') THEN
               IF(K < 33) THEN
                  O := CONV_INTEGER(DATA_IN);
                  IF(K = 1) THEN
                   -- INITIALIZATION
                             D(1) := B(O);
                             D(2) := B(32 + O) - 9999999;
                             D(3) := B(64 + O) - 9999999;
                             D(4) := B(96 + O) - 9999999;
                             D(5) := B(128 + O) - 9999999;
                             D(6) := B(160 + O) - 9999999;
                             K := K + 1;
                      ELSE
                      D(6) := FIND_MAX2((D(5)-1211),D(6)) + B(160+O);
                                          D(5) := FIND_MAX2((D(4)-1558),(D(5)-
354)) + B(128+O);
                                          D(4) := FIND_MAX2((D(3)-1662),(D(4)-
236)) + B(96+O);
                                          D(3) := FIND_MAX2((D(2)-1729),(D(3)-
210)) + B(64+O);
                                          D(2) := FIND_MAX2((D(1)-1899),(D(2)-
195)) + B(32+O);
                                          D(1) := D(1) - 162 + B(O);
                                          K := K + 1;
                      END IF;
                      ELSIF(K = 33) THEN
```

```
--            -- TERMINATION
                    LOG_LIK <= CONV_STD_LOGIC_VECTOR(FIND_MAX1(D),32);
                     MAX_ENABLE <= '1';
                    END IF;
                    END IF;
              END IF;
       END PROCESS PRO4;
    END VITERBI_ALGO4;

        ARCHITECTURE MAX_MODULE OF MAX_MODULE IS

    BEGIN
    PRO7 : PROCESS(MAX_ENABLE0, MAX_ENABLE1, MAX_ENABLE2, MAX_ENABLE3, MAX_ENABLE4)
    VARIABLE MAX : INTEGER;
        BEGIN
       IF ( MAX_ENABLE0 = '1' AND MAX_ENABLE1 = '1' AND MAX_ENABLE2 = '1' AND
MAX_ENABLE3 = '1' AND MAX_ENABLE4 = '1') THEN
                     MAX := CONV_INTEGER(LOG_LIK0);
                             IF ( CONV_INTEGER(LOG_LIK1) > MAX) THEN MAX :=
CONV_INTEGER(LOG_LIK1); END IF;
                             IF ( CONV_INTEGER(LOG_LIK2) > MAX) THEN MAX :=
CONV_INTEGER(LOG_LIK2); END IF;
                             IF ( CONV_INTEGER(LOG_LIK3) > MAX) THEN MAX :=
CONV_INTEGER(LOG_LIK3); END IF;
                             IF ( CONV_INTEGER(LOG_LIK4) > MAX) THEN MAX :=
CONV_INTEGER(LOG_LIK4); END IF;
                     IF (MAX = CONV_INTEGER(LOG_LIK0)) THEN RECOG_NO <= "0000";
                             ELSIF (MAX = CONV_INTEGER(LOG_LIK1)) THEN RECOG_NO <=
"0001";
                             ELSIF (MAX = CONV_INTEGER(LOG_LIK2)) THEN RECOG_NO <=
"0010";
                             ELSIF (MAX = CONV_INTEGER(LOG_LIK3)) THEN RECOG_NO <=
"0011";
                             ELSE RECOG_NO <= "0100";
                             END IF;
                    ELSE
            RECOG_NO <= "ZZZZ";
             END IF;
    END PROCESS PRO7;
END MAX_MODULE;
```

APPENDIX – C

SYNTHESIS REPORT

```
RELEASE 7.1I - XST H.38
COPYRIGHT (C) 1995-2005 XILINX, INC.  ALL RIGHTS RESERVED.
--> PARAMETER TMPDIR SET TO __PROJNAV
CPU : 0.00 / 1.53 S | ELAPSED : 0.00 / 1.00 S

--> PARAMETER XSTHDPDIR SET TO ./XST
CPU : 0.00 / 1.53 S | ELAPSED : 0.00 / 1.00 S

--> READING DESIGN: VITERBI_SIM.PRJ

TABLE OF CONTENTS
  1) SYNTHESIS OPTIONS SUMMARY
  2) HDL COMPILATION
  3) HDL ANALYSIS
  4) HDL SYNTHESIS
  5) ADVANCED HDL SYNTHESIS
     5.1) HDL SYNTHESIS REPORT
  6) LOW LEVEL SYNTHESIS
  7) FINAL REPORT
     7.1) DEVICE UTILIZATION SUMMARY
     7.2) TIMING REPORT


=========================================================================
==
*                     SYNTHESIS OPTIONS SUMMARY                         *
=========================================================================
==
---- SOURCE PARAMETERS
INPUT FILE NAME                    : "VITERBI_SIM.PRJ"
INPUT FORMAT                       : MIXED
IGNORE SYNTHESIS CONSTRAINT FILE   : NO

---- TARGET PARAMETERS
OUTPUT FILE NAME                   : "VITERBI_SIM"
OUTPUT FORMAT                      : NGC
TARGET DEVICE                      : XC2VP50-6-FF1148

---- SOURCE OPTIONS
TOP MODULE NAME                    : VITERBI_SIM
AUTOMATIC FSM EXTRACTION           : YES
FSM ENCODING ALGORITHM             : AUTO
FSM STYLE                          : LUT
RAM EXTRACTION                     : YES
RAM STYLE                          : AUTO
ROM EXTRACTION                     : YES
ROM STYLE                          : AUTO
MUX EXTRACTION                     : YES
DECODER EXTRACTION                 : YES
PRIORITY ENCODER EXTRACTION        : YES
SHIFT REGISTER EXTRACTION          : YES
LOGICAL SHIFTER EXTRACTION         : YES
```

```
XOR Collapsing                    : YES
Resource Sharing                  : YES
Multiplier Style                  : auto
Automatic Register Balancing      : No

---- Target Options
Add IO Buffers                    : YES
Global Maximum Fanout             : 500
Add Generic Clock Buffer(BUFG)     : 16
Register Duplication              : YES
Equivalent register Removal       : YES
Slice Packing                     : YES
Pack IO Registers into IOBs       : auto

---- General Options
Optimization Goal                 : Speed
Optimization Effort               : 1
Keep Hierarchy                    : NO
Global Optimization               : AllClockNets
RTL Output                        : Yes
Write Timing Constraints          : NO
Hierarchy Separator               : /
Bus Delimiter                     : <>
Case Specifier                    : maintain
Slice Utilization Ratio           : 100
Slice Utilization Ratio Delta     : 5

---- Other Options
lso                               : viterbi_sim.lso
Read Cores                        : YES
cross_clock_analysis              : NO
verilog2001                       : YES
safe_implementation               : No
Optimize Instantiated Primitives  : NO
tristate2logic                    : Yes
use_clock_enable                  : Yes
use_sync_set                      : Yes
use_sync_reset                    : Yes
enable_auto_floorplanning         : No

=========================================================================
==


=========================================================================
==
*                      HDL Compilation                                  *
=========================================================================
==
Compiling vhdl file "C:/adshfllf/123/viterbi_algo232.vhd" in Library work.
Architecture viterbi_sim of Entity viterbi_sim is up to date.
Architecture rom_module0 of Entity rom_module0 is up to date.
Architecture rom_module1 of Entity rom_module1 is up to date.
Architecture rom_module2 of Entity rom_module2 is up to date.
Architecture rom_module3 of Entity rom_module3 is up to date.
Architecture rom_module4 of Entity rom_module4 is up to date.
Architecture viterbi_algo0 of Entity viterbi_algo0 is up to date.
```

```
Architecture viterbi_algo1 of Entity viterbi_algo1 is up to date.
Architecture viterbi_algo2 of Entity viterbi_algo2 is up to date.
Architecture viterbi_algo3 of Entity viterbi_algo3 is up to date.
Architecture viterbi_algo4 of Entity viterbi_algo4 is up to date.
Architecture max_module of Entity max_module is up to date.

=========================================================================
==
*                          HDL Analysis                                 *
=========================================================================
==
Analyzing Entity <viterbi_sim> (Architecture <viterbi_sim>).
Entity <viterbi_sim> analyzed. Unit <viterbi_sim> generated.

Analyzing Entity <viterbi_algo0> (Architecture <viterbi_algo0>).
INFO:Xst:1304 - Contents of register <max_enable> in unit <viterbi_algo0> never
changes during circuit operation. The register is replaced by logic.
Entity <viterbi_algo0> analyzed. Unit <viterbi_algo0> generated.

Analyzing Entity <viterbi_algo1> (Architecture <viterbi_algo1>).
INFO:Xst:1304 - Contents of register <max_enable> in unit <viterbi_algo1> never
changes during circuit operation. The register is replaced by logic.
Entity <viterbi_algo1> analyzed. Unit <viterbi_algo1> generated.

Analyzing Entity <viterbi_algo2> (Architecture <viterbi_algo2>).
INFO:Xst:1304 - Contents of register <max_enable> in unit <viterbi_algo2> never
changes during circuit operation. The register is replaced by logic.
Entity <viterbi_algo2> analyzed. Unit <viterbi_algo2> generated.

Analyzing Entity <viterbi_algo3> (Architecture <viterbi_algo3>).
INFO:Xst:1304 - Contents of register <max_enable> in unit <viterbi_algo3> never
changes during circuit operation. The register is replaced by logic.
Entity <viterbi_algo3> analyzed. Unit <viterbi_algo3> generated.

Analyzing Entity <viterbi_algo4> (Architecture <viterbi_algo4>).
INFO:Xst:1304 - Contents of register <max_enable> in unit <viterbi_algo4> never
changes during circuit operation. The register is replaced by logic.
Entity <viterbi_algo4> analyzed. Unit <viterbi_algo4> generated.

Analyzing Entity <rom_module0> (Architecture <rom_module0>).
Entity <rom_module0> analyzed. Unit <rom_module0> generated.

Analyzing Entity <rom_module1> (Architecture <rom_module1>).
Entity <rom_module1> analyzed. Unit <rom_module1> generated.

Analyzing Entity <rom_module2> (Architecture <rom_module2>).
Entity <rom_module2> analyzed. Unit <rom_module2> generated.

Analyzing Entity <rom_module3> (Architecture <rom_module3>).
Entity <rom_module3> analyzed. Unit <rom_module3> generated.

Analyzing Entity <rom_module4> (Architecture <rom_module4>).
Entity <rom_module4> analyzed. Unit <rom_module4> generated.

Analyzing Entity <max_module> (Architecture <max_module>).
Entity <max_module> analyzed. Unit <max_module> generated.
```

```
=========================================================================
==
*                          HDL Synthesis                                *
=========================================================================
==

Synthesizing Unit <max_module>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Clock> is never used.
    Found 4-bit tristate buffer for signal <recog_no>.
    Found 32-bit comparator greater for signal <$n0015> created at line 1981.
    Found 32-bit comparator greater for signal <$n0016> created at line 1982.
    Found 32-bit comparator greater for signal <$n0017> created at line 1983.
    Found 32-bit comparator greater for signal <$n0018> created at line 1984.
    Found 32-bit comparator equal for signal <$n0019> created at line 1985.
    Found 32-bit comparator equal for signal <$n0020> created at line 1985.
    Found 32-bit comparator equal for signal <$n0021> created at line 1986.
    Found 32-bit comparator equal for signal <$n0022> created at line 1987.
    Summary:
       inferred   8 Comparator(s).
       inferred   4 Tristate(s).
Unit <max_module> synthesized.


Synthesizing Unit <rom_module4>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
    Found 32x8-bit ROM for signal <$n0002> created at line 635.
    Found 8-bit tristate buffer for signal <Data_out>.
    Found 1-bit register for signal <Data_ready>.
    Found 8-bit register for signal <Mtridata_Data_out> created at line 635.
    Found 1-bit register for signal <Mtrien_Data_out> created at line 635.
    Summary:
       inferred   1 ROM(s).
       inferred  10 D-type flip-flop(s).
       inferred   8 Tristate(s).
Unit <rom_module4> synthesized.


Synthesizing Unit <rom_module3>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
    Found 32x8-bit ROM for signal <$n0002> created at line 578.
    Found 8-bit tristate buffer for signal <Data_out>.
    Found 1-bit register for signal <Data_ready>.
    Found 8-bit register for signal <Mtridata_Data_out> created at line 578.
    Found 1-bit register for signal <Mtrien_Data_out> created at line 578.
    Summary:
       inferred   1 ROM(s).
       inferred  10 D-type flip-flop(s).
       inferred   8 Tristate(s).
Unit <rom_module3> synthesized.


Synthesizing Unit <rom_module2>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
    Found 32x8-bit ROM for signal <$n0002> created at line 521.
    Found 8-bit tristate buffer for signal <Data_out>.
```

```
    Found 1-bit register for signal <Data_ready>.
    Found 8-bit register for signal <Mtridata_Data_out> created at line 521.
    Found 1-bit register for signal <Mtrien_Data_out> created at line 521.
    Summary:
       inferred   1 ROM(s).
       inferred  10 D-type flip-flop(s).
       inferred   8 Tristate(s).
Unit <rom_module2> synthesized.


Synthesizing Unit <rom_module1>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
    Found 32x8-bit ROM for signal <$n0002> created at line 464.
    Found 8-bit tristate buffer for signal <Data_out>.
    Found 1-bit register for signal <Data_ready>.
    Found 8-bit register for signal <Mtridata_Data_out> created at line 464.
    Found 1-bit register for signal <Mtrien_Data_out> created at line 464.
    Summary:
       inferred   1 ROM(s).
       inferred  10 D-type flip-flop(s).
       inferred   8 Tristate(s).
Unit <rom_module1> synthesized.


Synthesizing Unit <rom_module0>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
    Found 32x8-bit ROM for signal <$n0002> created at line 407.
    Found 8-bit tristate buffer for signal <Data_out>.
    Found 1-bit register for signal <Data_ready>.
    Found 8-bit register for signal <Mtridata_Data_out> created at line 407.
    Found 1-bit register for signal <Mtrien_Data_out> created at line 407.
    Summary:
       inferred   1 ROM(s).
       inferred  10 D-type flip-flop(s).
       inferred   8 Tristate(s).
Unit <rom_module0> synthesized.


Synthesizing Unit <viterbi_algo4>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Data_in<7:6>> is never used.
    Found 64x82-bit ROM for signal <$n0082>.
    Found 33x25-bit ROM for signal <$n0007> created at line 1952.
    Found 5-bit tristate buffer for signal <Address>.
    Found 32-bit register for signal <log_lik>.
    Found 1-bit register for signal <Enable>.
    Found 32-bit subtractor for signal <$n0009> created at line 1917.
    Found 32-bit subtractor for signal <$n0010> created at line 1917.
    Found 32-bit subtractor for signal <$n0011> created at line 1917.
    Found 32-bit subtractor for signal <$n0012> created at line 1917.
    Found 26-bit subtractor for signal <$n0025> created at line 1952.
    Found 26-bit subtractor for signal <$n0026> created at line 1951.
    Found 25-bit subtractor for signal <$n0027> created at line 1950.
    Found 25-bit subtractor for signal <$n0028> created at line 1949.
    Found 25-bit subtractor for signal <$n0029> created at line 1948.
    Found 32-bit comparator less for signal <$n0040> created at line 1924.
    Found 32-bit subtractor for signal <$n0054> created at line 1957.
```

```
    Found 32-bit subtractor for signal <$n0055> created at line 1957.
    Found 32-bit subtractor for signal <$n0056> created at line 1958.
    Found 32-bit subtractor for signal <$n0057> created at line 1958.
    Found 32-bit subtractor for signal <$n0058> created at line 1959.
    Found 32-bit subtractor for signal <$n0059> created at line 1959.
    Found 32-bit subtractor for signal <$n0060> created at line 1956.
    Found 32-bit subtractor for signal <$n0061> created at line 1956.
    Found 32-bit subtractor for signal <$n0062> created at line 1955.
    Found 32-bit subtractor for signal <$n0063> created at line 1960.
    Found 32-bit adder for signal <$n0064> created at line 1960.
    Found 32-bit adder for signal <$n0065> created at line 1959.
    Found 32-bit adder for signal <$n0066> created at line 1958.
    Found 32-bit adder for signal <$n0067> created at line 1957.
    Found 32-bit adder for signal <$n0068> created at line 1956.
    Found 32-bit adder for signal <$n0069> created at line 1955.
    Found 32-bit comparator less for signal <$n0070> created at line 1911.
    Found 32-bit comparator greater for signal <$n0071> created at line 1917.
    Found 32-bit comparator less for signal <$n0072> created at line 1911.
    Found 32-bit comparator less for signal <$n0073> created at line 1911.
    Found 32-bit comparator less for signal <$n0074> created at line 1911.
    Found 32-bit comparator less for signal <$n0076> created at line 1911.
    Found 32-bit comparator greater for signal <$n0078> created at line 1917.
    Found 32-bit comparator greater for signal <$n0079> created at line 1917.
    Found 32-bit comparator greater for signal <$n0080> created at line 1917.
    Found 32-bit comparator greater for signal <$n0081> created at line 1917.
    Found 32-bit comparator greatequal for signal <$n0146> created at line 1943.
    Found 192-bit register for signal <D>.
    Found 32-bit up counter for signal <i>.
    Found 32-bit up counter for signal <k>.
    Found 5-bit register for signal <Mtridata_Address> created at line 1926.
    Found 1-bit register for signal <Mtrien_Address> created at line 1926.
    Summary:
        inferred   2 ROM(s).
        inferred   2 Counter(s).
        inferred 231 D-type flip-flop(s).
        inferred  25 Adder/Subtractor(s).
        inferred  12 Comparator(s).
        inferred   5 Tristate(s).
Unit <viterbi_algo4> synthesized.


Synthesizing Unit <viterbi_algo3>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Data_in<7:6>> is never used.
    Found 64x95-bit ROM for signal <$n0082>.
    Found 33x25-bit ROM for signal <$n0007> created at line 1685.
    Found 5-bit tristate buffer for signal <Address>.
    Found 32-bit register for signal <log_lik>.
    Found 1-bit register for signal <Enable>.
    Found 32-bit subtractor for signal <$n0009> created at line 1650.
    Found 32-bit subtractor for signal <$n0010> created at line 1650.
    Found 32-bit subtractor for signal <$n0011> created at line 1650.
    Found 32-bit subtractor for signal <$n0012> created at line 1650.
    Found 26-bit subtractor for signal <$n0025> created at line 1685.
    Found 26-bit subtractor for signal <$n0026> created at line 1684.
    Found 26-bit subtractor for signal <$n0027> created at line 1683.
    Found 26-bit subtractor for signal <$n0028> created at line 1682.
```

```
    Found 25-bit subtractor for signal <$n0029> created at line 1681.
    Found 32-bit comparator less for signal <$n0040> created at line 1657.
    Found 32-bit subtractor for signal <$n0054> created at line 1690.
    Found 32-bit subtractor for signal <$n0055> created at line 1690.
    Found 32-bit subtractor for signal <$n0056> created at line 1691.
    Found 32-bit subtractor for signal <$n0057> created at line 1691.
    Found 32-bit subtractor for signal <$n0058> created at line 1692.
    Found 32-bit subtractor for signal <$n0059> created at line 1692.
    Found 32-bit subtractor for signal <$n0060> created at line 1689.
    Found 32-bit subtractor for signal <$n0061> created at line 1689.
    Found 32-bit subtractor for signal <$n0062> created at line 1688.
    Found 32-bit subtractor for signal <$n0063> created at line 1693.
    Found 32-bit adder for signal <$n0064> created at line 1693.
    Found 32-bit adder for signal <$n0065> created at line 1692.
    Found 32-bit adder for signal <$n0066> created at line 1691.
    Found 32-bit adder for signal <$n0067> created at line 1690.
    Found 32-bit adder for signal <$n0068> created at line 1689.
    Found 32-bit adder for signal <$n0069> created at line 1688.
    Found 32-bit comparator less for signal <$n0070> created at line 1644.
    Found 32-bit comparator greater for signal <$n0071> created at line 1650.
    Found 32-bit comparator less for signal <$n0072> created at line 1644.
    Found 32-bit comparator less for signal <$n0073> created at line 1644.
    Found 32-bit comparator less for signal <$n0074> created at line 1644.
    Found 32-bit comparator less for signal <$n0076> created at line 1644.
    Found 32-bit comparator greater for signal <$n0078> created at line 1650.
    Found 32-bit comparator greater for signal <$n0079> created at line 1650.
    Found 32-bit comparator greater for signal <$n0080> created at line 1650.
    Found 32-bit comparator greater for signal <$n0081> created at line 1650.
    Found 32-bit comparator greatequal for signal <$n0146> created at line 1676.
    Found 192-bit register for signal <D>.
    Found 32-bit up counter for signal <i>.
    Found 32-bit up counter for signal <k>.
    Found 5-bit register for signal <Mtridata_Address> created at line 1659.
    Found 1-bit register for signal <Mtrien_Address> created at line 1659.
    Summary:
        inferred   2 ROM(s).
        inferred   2 Counter(s).
        inferred 231 D-type flip-flop(s).
        inferred  25 Adder/Subtractor(s).
        inferred  12 Comparator(s).
        inferred   5 Tristate(s).
Unit <viterbi_algo3> synthesized.


Synthesizing Unit <viterbi_algo2>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Data_in<7:6>> is never used.
    Found 64x84-bit ROM for signal <$n0082>.
    Found 33x25-bit ROM for signal <$n0007> created at line 1420.
    Found 5-bit tristate buffer for signal <Address>.
    Found 32-bit register for signal <log_lik>.
    Found 1-bit register for signal <Enable>.
    Found 32-bit subtractor for signal <$n0009> created at line 1385.
    Found 32-bit subtractor for signal <$n0010> created at line 1385.
    Found 32-bit subtractor for signal <$n0011> created at line 1385.
    Found 32-bit subtractor for signal <$n0012> created at line 1385.
    Found 26-bit subtractor for signal <$n0025> created at line 1420.
```

```
    Found 26-bit subtractor for signal <$n0026> created at line 1419.
    Found 25-bit subtractor for signal <$n0027> created at line 1418.
    Found 25-bit subtractor for signal <$n0028> created at line 1417.
    Found 25-bit subtractor for signal <$n0029> created at line 1416.
    Found 32-bit comparator less for signal <$n0040> created at line 1392.
    Found 32-bit subtractor for signal <$n0054> created at line 1425.
    Found 32-bit subtractor for signal <$n0055> created at line 1425.
    Found 32-bit subtractor for signal <$n0056> created at line 1426.
    Found 32-bit subtractor for signal <$n0057> created at line 1426.
    Found 32-bit subtractor for signal <$n0058> created at line 1427.
    Found 32-bit subtractor for signal <$n0059> created at line 1427.
    Found 32-bit subtractor for signal <$n0060> created at line 1424.
    Found 32-bit subtractor for signal <$n0061> created at line 1424.
    Found 32-bit subtractor for signal <$n0062> created at line 1423.
    Found 32-bit subtractor for signal <$n0063> created at line 1428.
    Found 32-bit adder for signal <$n0064> created at line 1428.
    Found 32-bit adder for signal <$n0065> created at line 1427.
    Found 32-bit adder for signal <$n0066> created at line 1426.
    Found 32-bit adder for signal <$n0067> created at line 1425.
    Found 32-bit adder for signal <$n0068> created at line 1424.
    Found 32-bit adder for signal <$n0069> created at line 1423.
    Found 32-bit comparator less for signal <$n0070> created at line 1379.
    Found 32-bit comparator greater for signal <$n0071> created at line 1385.
    Found 32-bit comparator less for signal <$n0072> created at line 1379.
    Found 32-bit comparator less for signal <$n0073> created at line 1379.
    Found 32-bit comparator less for signal <$n0074> created at line 1379.
    Found 32-bit comparator less for signal <$n0076> created at line 1379.
    Found 32-bit comparator greater for signal <$n0078> created at line 1385.
    Found 32-bit comparator greater for signal <$n0079> created at line 1385.
    Found 32-bit comparator greater for signal <$n0080> created at line 1385.
    Found 32-bit comparator greater for signal <$n0081> created at line 1385.
    Found 32-bit comparator greatequal for signal <$n0146> created at line 1411.
    Found 192-bit register for signal <D>.
    Found 32-bit up counter for signal <i>.
    Found 32-bit up counter for signal <k>.
    Found 5-bit register for signal <Mtridata_Address> created at line 1394.
    Found 1-bit register for signal <Mtrien_Address> created at line 1394.
    Summary:
        inferred   2 ROM(s).
        inferred   2 Counter(s).
        inferred 231 D-type flip-flop(s).
        inferred  25 Adder/Subtractor(s).
        inferred  12 Comparator(s).
        inferred   5 Tristate(s).
Unit <viterbi_algo2> synthesized.


Synthesizing Unit <viterbi_algo1>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Data_in<7:6>> is never used.
    Found 64x83-bit ROM for signal <$n0082>.
    Found 33x25-bit ROM for signal <$n0007> created at line 1153.
    Found 5-bit tristate buffer for signal <Address>.
    Found 32-bit register for signal <log_lik>.
    Found 1-bit register for signal <Enable>.
    Found 32-bit subtractor for signal <$n0009> created at line 1118.
    Found 32-bit subtractor for signal <$n0010> created at line 1118.
```

```
    Found 32-bit subtractor for signal <$n0011> created at line 1118.
    Found 32-bit subtractor for signal <$n0012> created at line 1118.
    Found 26-bit subtractor for signal <$n0025> created at line 1153.
    Found 26-bit subtractor for signal <$n0026> created at line 1152.
    Found 25-bit subtractor for signal <$n0027> created at line 1151.
    Found 25-bit subtractor for signal <$n0028> created at line 1150.
    Found 25-bit subtractor for signal <$n0029> created at line 1149.
    Found 32-bit comparator less for signal <$n0040> created at line 1125.
    Found 32-bit subtractor for signal <$n0054> created at line 1158.
    Found 32-bit subtractor for signal <$n0055> created at line 1158.
    Found 32-bit subtractor for signal <$n0056> created at line 1159.
    Found 32-bit subtractor for signal <$n0057> created at line 1159.
    Found 32-bit subtractor for signal <$n0058> created at line 1160.
    Found 32-bit subtractor for signal <$n0059> created at line 1160.
    Found 32-bit subtractor for signal <$n0060> created at line 1157.
    Found 32-bit subtractor for signal <$n0061> created at line 1157.
    Found 32-bit subtractor for signal <$n0062> created at line 1156.
    Found 32-bit subtractor for signal <$n0063> created at line 1161.
    Found 32-bit adder for signal <$n0064> created at line 1161.
    Found 32-bit adder for signal <$n0065> created at line 1160.
    Found 32-bit adder for signal <$n0066> created at line 1159.
    Found 32-bit adder for signal <$n0067> created at line 1158.
    Found 32-bit adder for signal <$n0068> created at line 1157.
    Found 32-bit adder for signal <$n0069> created at line 1156.
    Found 32-bit comparator less for signal <$n0070> created at line 1112.
    Found 32-bit comparator greater for signal <$n0071> created at line 1118.
    Found 32-bit comparator less for signal <$n0072> created at line 1112.
    Found 32-bit comparator less for signal <$n0073> created at line 1112.
    Found 32-bit comparator less for signal <$n0074> created at line 1112.
    Found 32-bit comparator less for signal <$n0076> created at line 1112.
    Found 32-bit comparator greater for signal <$n0078> created at line 1118.
    Found 32-bit comparator greater for signal <$n0079> created at line 1118.
    Found 32-bit comparator greater for signal <$n0080> created at line 1118.
    Found 32-bit comparator greater for signal <$n0081> created at line 1118.
    Found 32-bit comparator greatequal for signal <$n0146> created at line 1144.
    Found 192-bit register for signal <D>.
    Found 32-bit up counter for signal <i>.
    Found 32-bit up counter for signal <k>.
    Found 5-bit register for signal <Mtridata_Address> created at line 1127.
    Found 1-bit register for signal <Mtrien_Address> created at line 1127.
    Summary:
        inferred   2 ROM(s).
        inferred   2 Counter(s).
        inferred 231 D-type flip-flop(s).
        inferred  25 Adder/Subtractor(s).
        inferred  12 Comparator(s).
        inferred   5 Tristate(s).
Unit <viterbi_algo1> synthesized.


Synthesizing Unit <viterbi_algo0>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
WARNING:Xst:647 - Input <Data_in<7:6>> is never used.
    Found 64x100-bit ROM for signal <$n0082>.
    Found 33x25-bit ROM for signal <$n0007> created at line 888.
    Found 5-bit tristate buffer for signal <Address>.
    Found 32-bit register for signal <log_lik>.
```

```
    Found 1-bit register for signal <Enable>.
    Found 32-bit subtractor for signal <$n0009> created at line 854.
    Found 32-bit subtractor for signal <$n0010> created at line 854.
    Found 32-bit subtractor for signal <$n0011> created at line 854.
    Found 32-bit subtractor for signal <$n0012> created at line 854.
    Found 26-bit subtractor for signal <$n0025> created at line 888.
    Found 26-bit subtractor for signal <$n0026> created at line 887.
    Found 26-bit subtractor for signal <$n0027> created at line 886.
    Found 26-bit subtractor for signal <$n0028> created at line 885.
    Found 26-bit subtractor for signal <$n0029> created at line 884.
    Found 32-bit comparator less for signal <$n0040> created at line 861.
    Found 32-bit subtractor for signal <$n0054> created at line 893.
    Found 32-bit subtractor for signal <$n0055> created at line 893.
    Found 32-bit subtractor for signal <$n0056> created at line 894.
    Found 32-bit subtractor for signal <$n0057> created at line 894.
    Found 32-bit subtractor for signal <$n0058> created at line 895.
    Found 32-bit subtractor for signal <$n0059> created at line 895.
    Found 32-bit subtractor for signal <$n0060> created at line 892.
    Found 32-bit subtractor for signal <$n0061> created at line 892.
    Found 32-bit subtractor for signal <$n0062> created at line 891.
    Found 32-bit subtractor for signal <$n0063> created at line 896.
    Found 32-bit adder for signal <$n0064> created at line 896.
    Found 32-bit adder for signal <$n0065> created at line 895.
    Found 32-bit adder for signal <$n0066> created at line 894.
    Found 32-bit adder for signal <$n0067> created at line 893.
    Found 32-bit adder for signal <$n0068> created at line 892.
    Found 32-bit adder for signal <$n0069> created at line 891.
    Found 32-bit comparator less for signal <$n0070> created at line 848.
    Found 32-bit comparator greater for signal <$n0071> created at line 854.
    Found 32-bit comparator less for signal <$n0072> created at line 848.
    Found 32-bit comparator less for signal <$n0073> created at line 848.
    Found 32-bit comparator less for signal <$n0074> created at line 848.
    Found 32-bit comparator less for signal <$n0076> created at line 848.
    Found 32-bit comparator greater for signal <$n0078> created at line 854.
    Found 32-bit comparator greater for signal <$n0079> created at line 854.
    Found 32-bit comparator greater for signal <$n0080> created at line 854.
    Found 32-bit comparator greater for signal <$n0081> created at line 854.
    Found 32-bit comparator greatequal for signal <$n0146> created at line 879.
    Found 192-bit register for signal <D>.
    Found 32-bit up counter for signal <i>.
    Found 32-bit up counter for signal <k>.
    Found 5-bit register for signal <Mtridata_Address> created at line 863.
    Found 1-bit register for signal <Mtrien_Address> created at line 863.
    Summary:
       inferred   2 ROM(s).
       inferred   2 Counter(s).
       inferred 231 D-type flip-flop(s).
       inferred  25 Adder/Subtractor(s).
       inferred  12 Comparator(s).
       inferred   5 Tristate(s).
Unit <viterbi_algo0> synthesized.


Synthesizing Unit <viterbi_sim>.
    Related source file is "C:/adshfllf/123/viterbi_algo232.vhd".
Unit <viterbi_sim> synthesized.
```

```
INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic
operations in this design can share the same physical resources for reduced device
utilization. For improved clock frequency you may try to disable resource sharing.

=========================================================================
==
*                     Advanced HDL Synthesis                            *
=========================================================================
==

Advanced RAM inference ...
INFO:Xst:1647 - Data output of ROM <Mrom__n0002> in block <rom_module0> is tied to
register <Mtridata_Data_out> in block <rom_module0>.
INFO:Xst:1650 - The register is removed and the ROM is implemented as read-only block
RAM.
INFO:Xst:1647 - Data output of ROM <Mrom__n0002> in block <rom_module1> is tied to
register <Mtridata_Data_out> in block <rom_module1>.
INFO:Xst:1650 - The register is removed and the ROM is implemented as read-only block
RAM.
INFO:Xst:1647 - Data output of ROM <Mrom__n0002> in block <rom_module2> is tied to
register <Mtridata_Data_out> in block <rom_module2>.
INFO:Xst:1650 - The register is removed and the ROM is implemented as read-only block
RAM.
INFO:Xst:1647 - Data output of ROM <Mrom__n0002> in block <rom_module3> is tied to
register <Mtridata_Data_out> in block <rom_module3>.
INFO:Xst:1650 - The register is removed and the ROM is implemented as read-only block
RAM.
INFO:Xst:1647 - Data output of ROM <Mrom__n0002> in block <rom_module4> is tied to
register <Mtridata_Data_out> in block <rom_module4>.
INFO:Xst:1650 - The register is removed and the ROM is implemented as read-only block
RAM.
Advanced multiplier inference ...
Advanced Registered AddSub inference ...
Dynamic shift register inference ...

=========================================================================
==
HDL Synthesis Report

Macro Statistics
# Block RAMs                       : 5
 32x8-bit single-port block RAM    : 5
# ROMs                             : 10
 33x25-bit ROM                     : 5
 64x100-bit ROM                    : 1
 64x82-bit ROM                     : 1
 64x83-bit ROM                     : 1
 64x84-bit ROM                     : 1
 64x95-bit ROM                     : 1
# Adders/Subtractors               : 125
 25-bit subtractor                 : 10
 26-bit subtractor                 : 15
 32-bit adder                      : 30
 32-bit subtractor                 : 70
# Counters                         : 10
 32-bit up counter                 : 10
# Registers                        : 60
```

```
 1-BIT REGISTER                    : 20
 32-BIT REGISTER                   : 35
 5-BIT REGISTER                    : 5
# COMPARATORS                      : 68
 32-BIT COMPARATOR EQUAL           : 4
 32-BIT COMPARATOR GREATEQUAL      : 5
 32-BIT COMPARATOR GREATER         : 29
 32-BIT COMPARATOR LESS            : 30
# TRISTATES                        : 11
 4-BIT TRISTATE BUFFER             : 1
 5-BIT TRISTATE BUFFER             : 5
 8-BIT TRISTATE BUFFER             : 5

=========================================================================
==

=========================================================================
==
*                      LOW LEVEL SYNTHESIS                              *
=========================================================================
==
WARNING:XST:2042 - UNIT VITERBI_SIM: 40 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): W_DATA0<0>, W_DATA0<1>, W_DATA0<2>, W_DATA0<3>, W_DATA0<4>,
W_DATA0<5>, W_DATA0<6>, W_DATA0<7>, W_DATA1<0>, W_DATA1<1>, W_DATA1<2>,
W_DATA1<3>, W_DATA1<4>, W_DATA1<5>, W_DATA1<6>, W_DATA1<7>, W_DATA2<0>,
W_DATA2<1>, W_DATA2<2>, W_DATA2<3>, W_DATA2<4>, W_DATA2<5>, W_DATA2<6>,
W_DATA2<7>, W_DATA3<0>, W_DATA3<1>, W_DATA3<2>, W_DATA3<3>, W_DATA3<4>,
W_DATA3<5>, W_DATA3<6>, W_DATA3<7>, W_DATA4<0>, W_DATA4<1>, W_DATA4<2>,
W_DATA4<3>, W_DATA4<4>, W_DATA4<5>, W_DATA4<6>, W_DATA4<7>.
WARNING:XST:2042 - UNIT VITERBI_ALGO0: 5 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): ADDRESS<0>, ADDRESS<1>, ADDRESS<2>, ADDRESS<3>, ADDRESS<4>.
WARNING:XST:2042 - UNIT VITERBI_ALGO1: 5 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): ADDRESS<0>, ADDRESS<1>, ADDRESS<2>, ADDRESS<3>, ADDRESS<4>.
WARNING:XST:2042 - UNIT VITERBI_ALGO2: 5 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): ADDRESS<0>, ADDRESS<1>, ADDRESS<2>, ADDRESS<3>, ADDRESS<4>.
WARNING:XST:2042 - UNIT VITERBI_ALGO3: 5 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): ADDRESS<0>, ADDRESS<1>, ADDRESS<2>, ADDRESS<3>, ADDRESS<4>.
WARNING:XST:2042 - UNIT VITERBI_ALGO4: 5 INTERNAL TRISTATES ARE REPLACED BY LOGIC
(PULL-UP YES): ADDRESS<0>, ADDRESS<1>, ADDRESS<2>, ADDRESS<3>, ADDRESS<4>.

OPTIMIZING UNIT <VITERBI_SIM> ...

OPTIMIZING UNIT <VITERBI_ALGO4> ...

OPTIMIZING UNIT <VITERBI_ALGO3> ...

OPTIMIZING UNIT <VITERBI_ALGO2> ...

OPTIMIZING UNIT <VITERBI_ALGO1> ...

OPTIMIZING UNIT <VITERBI_ALGO0> ...
LOADING DEVICE FOR APPLICATION RF_DEVICE FROM FILE '2VP50.NPH' IN ENVIRONMENT
C:/XILINX.

MAPPING ALL EQUATIONS...
BUILDING AND OPTIMIZING FINAL NETLIST ...
FOUND AREA CONSTRAINT RATIO OF 100 (+ 5) ON BLOCK VITERBI_SIM, ACTUAL RATIO IS 24.
```

```
=========================================================================
==
*                          Final Report                                 *
=========================================================================
==
Final Results
RTL Top Level Output File Name     : viterbi_sim.ngr
Top Level Output File Name         : viterbi_sim
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 5

Macro Statistics :
# RAM                              : 5
#      32x8-bit single-port block RAM: 5
# ROMs                             : 10
#      33x25-bit ROM                : 5
#      64x100-bit ROM               : 1
#      64x82-bit ROM                : 1
#      64x83-bit ROM                : 1
#      64x84-bit ROM                : 1
#      64x95-bit ROM                : 1
# Registers                        : 70
#      1-bit register               : 20
#      32-bit register              : 45
#      5-bit register               : 5
# Tristates                        : 11
#      4-bit tristate buffer        : 1
#      5-bit tristate buffer        : 5
#      8-bit tristate buffer        : 5
# Adders/Subtractors               : 135
#      25-bit subtractor            : 10
#      26-bit subtractor            : 15
#      32-bit adder                 : 40
#      32-bit subtractor            : 70
# Comparators                      : 68
#      32-bit comparator equal      : 4
#      32-bit comparator greatequal: 5
#      32-bit comparator greater    : 29
#      32-bit comparator less       : 30

Cell Usage :
# BELS                             : 21451
#      GND                         : 1
#      INV                         : 1531
#      LUT1                        : 734
#      LUT1_L                      : 367
#      LUT2                        : 501
#      LUT2_L                      : 1190
#      LUT3                        : 1420
#      LUT3_D                      : 310
#      LUT3_L                      : 387
#      LUT4                        : 3023
```

```
#      LUT4_L                    : 780
#      MUXCY                     : 5782
#      MUXF5                     : 993
#      MUXF6                     : 472
#      VCC                       : 1
#      XORCY                     : 3959
# FlipFlops/Latches              : 1485
#      FD                        : 25
#      FDE                       : 1440
#      FDR                       : 20
# RAMS                           : 5
#      RAMB16_S36                : 5
# Clock Buffers                  : 1
#      BUFGP                     : 1
# IO Buffers                     : 4
#      OBUF                      : 4
=========================================================================
==

Device utilization summary:
---------------------------

Selected Device : 2vp50ff1148-6

 Number of Slices:                  5370  out of  23616    22%
 Number of Slice Flip Flops:        1485  out of  47232     3%
 Number of 4 input LUTs:            8712  out of  47232    18%
 Number of bonded IOBs:                5  out of    812     0%
 Number of BRAMs:                      5  out of    232     2%
 Number of GCLKs:                      1  out of     16     6%


=========================================================================
==
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
Clock                              | BUFGP                  | 1490  |
-----------------------------------+------------------------+-------+

Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 21.675ns (Maximum Frequency: 46.135MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 24.162ns
   Maximum combinational path delay: No path found
```

```
Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
==
Timing constraint: Default period analysis for Clock 'Clock'
  Clock period: 21.675ns (frequency: 46.135MHz)
  Total number of paths / destination ports: 264934090128 / 2950
-------------------------------------------------------------------------
--
Delay:               21.675ns (Levels of Logic = 170)
  Source:            VA4/D_1_0 (FF)
  Destination:       VA4/log_lik_31 (FF)
  Source Clock:      Clock rising
  Destination Clock: Clock rising

  Data Path: VA4/D_1_0 to VA4/log_lik_31
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q             6   0.374   0.667  VA4/D_1_0 (VA4/D_1_0)
     LUT2_L:I0->LO        1   0.313   0.000  VA4/XNor_stageLUT124 (VA4/N170)
     MUXCY:S->O           1   0.377   0.000  VA4/XNor_stageCY_RN_123
(VA4/XNor_stage_cyo124)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_124
(VA4/XNor_stage_cyo125)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_125
(VA4/XNor_stage_cyo126)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_126
(VA4/XNor_stage_cyo127)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_127
(VA4/XNor_stage_cyo128)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_128
(VA4/XNor_stage_cyo129)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_129
(VA4/XNor_stage_cyo130)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_130
(VA4/XNor_stage_cyo131)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_131
(VA4/XNor_stage_cyo132)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_132
(VA4/XNor_stage_cyo133)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_133
(VA4/XNor_stage_cyo134)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_134
(VA4/XNor_stage_cyo135)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_135
(VA4/XNor_stage_cyo136)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_136
(VA4/XNor_stage_cyo137)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_137
(VA4/XNor_stage_cyo138)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_138
(VA4/XNor_stage_cyo139)
     MUXCY:CI->O          1   0.042   0.000  VA4/XNor_stageCY_RN_139
(VA4/XNor_stage_cyo140)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_140
(VA4/XNOR_STAGE_CYO141)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_141
(VA4/XNOR_STAGE_CYO142)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_142
(VA4/XNOR_STAGE_CYO143)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_143
(VA4/XNOR_STAGE_CYO144)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_144
(VA4/XNOR_STAGE_CYO145)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_145
(VA4/XNOR_STAGE_CYO146)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_146
(VA4/XNOR_STAGE_CYO147)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_147
(VA4/XNOR_STAGE_CYO148)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_148
(VA4/XNOR_STAGE_CYO149)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_149
(VA4/XNOR_STAGE_CYO150)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_150
(VA4/XNOR_STAGE_CYO151)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_151
(VA4/XNOR_STAGE_CYO152)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_152
(VA4/XNOR_STAGE_CYO153)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_153
(VA4/XNOR_STAGE_CYO154)
    MUXCY:CI->O          63   0.524   0.870  VA4/GE_STAGECY_RN_4
(VA4/GE_STAGE_CYO4)
    LUT3:I2->O            2   0.313   0.588  VA4/_N0013<0>1 (VA4/_N0013<0>)
    LUT2_L:I0->LO         1   0.313   0.000  VA4/XNOR_STAGELUT31 (VA4/N71)
    MUXCY:S->O            1   0.377   0.000  VA4/XNOR_STAGECY_RN_30
(VA4/XNOR_STAGE_CYO31)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_31
(VA4/XNOR_STAGE_CYO32)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_32
(VA4/XNOR_STAGE_CYO33)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_33
(VA4/XNOR_STAGE_CYO34)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_34
(VA4/XNOR_STAGE_CYO35)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_35
(VA4/XNOR_STAGE_CYO36)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_36
(VA4/XNOR_STAGE_CYO37)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_37
(VA4/XNOR_STAGE_CYO38)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_38
(VA4/XNOR_STAGE_CYO39)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_39
(VA4/XNOR_STAGE_CYO40)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_40
(VA4/XNOR_STAGE_CYO41)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_41
(VA4/XNOR_STAGE_CYO42)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_42
(VA4/XNOR_STAGE_CYO43)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_43
(VA4/XNOR_STAGE_CYO44)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_44
(VA4/XNOR_STAGE_CYO45)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_45
(VA4/XNOR_STAGE_CYO46)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_46
(VA4/XNOR_STAGE_CYO47)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_47
(VA4/XNOR_STAGE_CYO48)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_48
(VA4/XNOR_STAGE_CYO49)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_49
(VA4/XNOR_STAGE_CYO50)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_50
(VA4/XNOR_STAGE_CYO51)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_51
(VA4/XNOR_STAGE_CYO52)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_52
(VA4/XNOR_STAGE_CYO53)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_53
(VA4/XNOR_STAGE_CYO54)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_54
(VA4/XNOR_STAGE_CYO55)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_55
(VA4/XNOR_STAGE_CYO56)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_56
(VA4/XNOR_STAGE_CYO57)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_57
(VA4/XNOR_STAGE_CYO58)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_58
(VA4/XNOR_STAGE_CYO59)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_59
(VA4/XNOR_STAGE_CYO60)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_60
(VA4/XNOR_STAGE_CYO61)
    MUXCY:CI->O          63   0.524   0.870  VA4/GE_STAGECY_RN_1
(VA4/GE_STAGE_CYO1)
    LUT3_D:I2->O          1   0.313   0.533  VA4/_N0014<0>1 (VA4/_N0014<0>)
    LUT2_L:I0->LO         1   0.313   0.000  VA4/XNOR_STAGELUT93 (VA4/N137)
    MUXCY:S->O            1   0.377   0.000  VA4/XNOR_STAGECY_RN_92
(VA4/XNOR_STAGE_CYO93)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_93
(VA4/XNOR_STAGE_CYO94)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_94
(VA4/XNOR_STAGE_CYO95)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_95
(VA4/XNOR_STAGE_CYO96)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_96
(VA4/XNOR_STAGE_CYO97)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_97
(VA4/XNOR_STAGE_CYO98)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_98
(VA4/XNOR_STAGE_CYO99)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_99
(VA4/XNOR_STAGE_CYO100)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_100
(VA4/XNOR_STAGE_CYO101)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_101
(VA4/XNOR_STAGE_CYO102)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_102
(VA4/XNOR_STAGE_CYO103)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_103
(VA4/XNOR_STAGE_CYO104)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_104
(VA4/XNOR_STAGE_CYO105)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_105
(VA4/XNOR_STAGE_CYO106)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_106
(VA4/XNOR_STAGE_CYO107)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_107
(VA4/XNOR_STAGE_CYO108)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_108
(VA4/XNOR_STAGE_CYO109)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_109
(VA4/XNOR_STAGE_CYO110)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_110
(VA4/XNOR_STAGE_CYO111)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_111
(VA4/XNOR_STAGE_CYO112)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_112
(VA4/XNOR_STAGE_CYO113)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_113
(VA4/XNOR_STAGE_CYO114)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_114
(VA4/XNOR_STAGE_CYO115)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_115
(VA4/XNOR_STAGE_CYO116)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_116
(VA4/XNOR_STAGE_CYO117)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_117
(VA4/XNOR_STAGE_CYO118)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_118
(VA4/XNOR_STAGE_CYO119)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_119
(VA4/XNOR_STAGE_CYO120)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_120
(VA4/XNOR_STAGE_CYO121)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_121
(VA4/XNOR_STAGE_CYO122)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_122
(VA4/XNOR_STAGE_CYO123)
    MUXCY:CI->O          63   0.524   0.870  VA4/GE_STAGECY_RN_3
(VA4/GE_STAGE_CYO3)
    LUT3:I2->O            2   0.313   0.588  VA4/_N0015<0>1 (VA4/_N0015<0>)
    LUT2_L:I0->LO         1   0.313   0.000  VA4/XNOR_STAGELUT62 (VA4/N104)
    MUXCY:S->O            1   0.377   0.000  VA4/XNOR_STAGECY_RN_61
(VA4/XNOR_STAGE_CYO62)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_62
(VA4/XNOR_STAGE_CYO63)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_63
(VA4/XNOR_STAGE_CYO64)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_64
(VA4/XNOR_STAGE_CYO65)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_65
(VA4/XNOR_STAGE_CYO66)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_66
(VA4/XNOR_STAGE_CYO67)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_67
(VA4/XNOR_STAGE_CYO68)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_68
(VA4/XNOR_STAGE_CYO69)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_69
(VA4/XNOR_STAGE_CYO70)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_70
(VA4/XNOR_STAGE_CYO71)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_71
(VA4/XNOR_STAGE_CYO72)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_72
(VA4/XNOR_STAGE_CYO73)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_73
(VA4/XNOR_STAGE_CYO74)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_74
(VA4/XNOR_STAGE_CYO75)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_75
(VA4/XNOR_STAGE_CYO76)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_76
(VA4/XNOR_STAGE_CYO77)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_77
(VA4/XNOR_STAGE_CYO78)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_78
(VA4/XNOR_STAGE_CYO79)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_79
(VA4/XNOR_STAGE_CYO80)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_80
(VA4/XNOR_STAGE_CYO81)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_81
(VA4/XNOR_STAGE_CYO82)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_82
(VA4/XNOR_STAGE_CYO83)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_83
(VA4/XNOR_STAGE_CYO84)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_84
(VA4/XNOR_STAGE_CYO85)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_85
(VA4/XNOR_STAGE_CYO86)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_86
(VA4/XNOR_STAGE_CYO87)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_87
(VA4/XNOR_STAGE_CYO88)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_88
(VA4/XNOR_STAGE_CYO89)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_89
(VA4/XNOR_STAGE_CYO90)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_90
(VA4/XNOR_STAGE_CYO91)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_91
(VA4/XNOR_STAGE_CYO92)
    MUXCY:CI->O          63   0.524   0.870  VA4/GE_STAGECY_RN_2
(VA4/GE_STAGE_CYO2)
    LUT3_D:I2->O          1   0.313   0.533  VA4/_N0016<0>1 (VA4/_N0016<0>)
    LUT2_L:I0->LO         1   0.313   0.000  VA4/XNOR_STAGELUT (VA4/N38)
    MUXCY:S->O            1   0.377   0.000  VA4/XNOR_STAGECY
(VA4/XNOR_STAGE_CYO)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_0
(VA4/XNOR_STAGE_CYO1)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_1
(VA4/XNOR_STAGE_CYO2)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_2
(VA4/XNOR_STAGE_CYO3)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_3
(VA4/XNOR_STAGE_CYO4)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_4
(VA4/XNOR_STAGE_CYO5)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_5
(VA4/XNOR_STAGE_CYO6)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_6
(VA4/XNOR_STAGE_CYO7)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_7
(VA4/XNOR_STAGE_CYO8)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_8
(VA4/XNOR_STAGE_CYO9)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_9
(VA4/XNOR_STAGE_CYO10)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_10
(VA4/XNOR_STAGE_CYO11)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_11
(VA4/XNOR_STAGE_CYO12)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_12
(VA4/XNOR_STAGE_CYO13)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_13
(VA4/XNOR_STAGE_CYO14)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_14
(VA4/XNOR_STAGE_CYO15)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_15
(VA4/XNOR_STAGE_CYO16)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_16
(VA4/XNOR_STAGE_CYO17)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_17
(VA4/XNOR_STAGE_CYO18)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_18
(VA4/XNOR_STAGE_CYO19)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_19
(VA4/XNOR_STAGE_CYO20)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_20
(VA4/XNOR_STAGE_CYO21)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_21
(VA4/XNOR_STAGE_CYO22)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_22
(VA4/XNOR_STAGE_CYO23)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_23
(VA4/XNOR_STAGE_CYO24)
```

```
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_24
(VA4/XNOR_STAGE_CYO25)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_25
(VA4/XNOR_STAGE_CYO26)
    MUXCY:CI->O           1   0.042   0.000  VA4/XNOR_STAGECY_RN_26
(VA4/XNOR_STAGE_CYO27)
    MUXCY:CI->O           1   0.041   0.000  VA4/XNOR_STAGECY_RN_27
(VA4/XNOR_STAGE_CYO28)
    MUXCY:CI->O           1   0.041   0.000  VA4/XNOR_STAGECY_RN_28
(VA4/XNOR_STAGE_CYO29)
    MUXCY:CI->O           1   0.041   0.000  VA4/XNOR_STAGECY_RN_29
(VA4/XNOR_STAGE_CYO30)
    MUXCY:CI->O          32   0.525   0.818  VA4/GE_STAGECY_RN_0
(VA4/GE_STAGE_CYO)
    LUT3_L:I2->LO         1   0.313   0.000  VA4/_N0032<3>1 (VA4/_N0032<3>)
    FDE:D                     0.234          VA4/LOG_LIK_3
    ----------------------------------------
    TOTAL                    21.675NS (14.471NS LOGIC, 7.205NS ROUTE)
                                    (66.8% LOGIC, 33.2% ROUTE)

=========================================================================
==
TIMING CONSTRAINT: DEFAULT OFFSET OUT AFTER FOR CLOCK 'CLOCK'
  TOTAL NUMBER OF PATHS / DESTINATION PORTS: 9919329408 / 3
-------------------------------------------------------------------------
--
OFFSET:              24.162NS (LEVELS OF LOGIC = 155)
  SOURCE:            VA0/LOG_LIK_0 (FF)
  DESTINATION:       RECOG_NO<2> (PAD)
  SOURCE CLOCK:      CLOCK RISING

  DATA PATH: VA0/LOG_LIK_0 TO RECOG_NO<2>
                            GATE     NET
    CELL:IN->OUT     FANOUT   DELAY   DELAY  LOGICAL NAME (NET NAME)
    ----------------------------------------  ------------
     FDE:C->Q             3   0.374   0.610  VA0/LOG_LIK_0 (VA0/LOG_LIK_0)
     LUT2:I0->O           1   0.313   0.000  XNOR_STAGELUT (N2)
     MUXCY:S->O           1   0.377   0.000  XNOR_STAGECY (XNOR_STAGE_CYO)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_0 (XNOR_STAGE_CYO1)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_1 (XNOR_STAGE_CYO2)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_2 (XNOR_STAGE_CYO3)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_3 (XNOR_STAGE_CYO4)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_4 (XNOR_STAGE_CYO5)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_5 (XNOR_STAGE_CYO6)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_6 (XNOR_STAGE_CYO7)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_7 (XNOR_STAGE_CYO8)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_8 (XNOR_STAGE_CYO9)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_9
(XNOR_STAGE_CYO10)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_10
(XNOR_STAGE_CYO11)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_11
(XNOR_STAGE_CYO12)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_12
(XNOR_STAGE_CYO13)
     MUXCY:CI->O          1   0.042   0.000  XNOR_STAGECY_RN_13
(XNOR_STAGE_CYO14)
```

```
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_14
(XNOR_STAGE_CYO15)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_15
(XNOR_STAGE_CYO16)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_16
(XNOR_STAGE_CYO17)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_17
(XNOR_STAGE_CYO18)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_18
(XNOR_STAGE_CYO19)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_19
(XNOR_STAGE_CYO20)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_20
(XNOR_STAGE_CYO21)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_21
(XNOR_STAGE_CYO22)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_22
(XNOR_STAGE_CYO23)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_23
(XNOR_STAGE_CYO24)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_24
(XNOR_STAGE_CYO25)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_25
(XNOR_STAGE_CYO26)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_26
(XNOR_STAGE_CYO27)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_27
(XNOR_STAGE_CYO28)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_28
(XNOR_STAGE_CYO29)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_29
(XNOR_STAGE_CYO30)
    MUXCY:CI->O          33   0.524   0.934  GE_STAGECY (GE_STAGE_CYO)
    LUT3:I0->O            2   0.313   0.588  MAX_FIND/_N0003<0>1
(MAX_FIND/_N0003<0>)
    LUT2:I0->O            1   0.313   0.000  XNOR_STAGELUT31 (N35)
    MUXCY:S->O            1   0.377   0.000  XNOR_STAGECY_RN_30
(XNOR_STAGE_CYO31)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_31
(XNOR_STAGE_CYO32)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_32
(XNOR_STAGE_CYO33)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_33
(XNOR_STAGE_CYO34)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_34
(XNOR_STAGE_CYO35)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_35
(XNOR_STAGE_CYO36)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_36
(XNOR_STAGE_CYO37)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_37
(XNOR_STAGE_CYO38)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_38
(XNOR_STAGE_CYO39)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_39
(XNOR_STAGE_CYO40)
```

```
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_40
(XNOR_STAGE_CYO41)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_41
(XNOR_STAGE_CYO42)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_42
(XNOR_STAGE_CYO43)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_43
(XNOR_STAGE_CYO44)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_44
(XNOR_STAGE_CYO45)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_45
(XNOR_STAGE_CYO46)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_46
(XNOR_STAGE_CYO47)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_47
(XNOR_STAGE_CYO48)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_48
(XNOR_STAGE_CYO49)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_49
(XNOR_STAGE_CYO50)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_50
(XNOR_STAGE_CYO51)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_51
(XNOR_STAGE_CYO52)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_52
(XNOR_STAGE_CYO53)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_53
(XNOR_STAGE_CYO54)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_54
(XNOR_STAGE_CYO55)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_55
(XNOR_STAGE_CYO56)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_56
(XNOR_STAGE_CYO57)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_57
(XNOR_STAGE_CYO58)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_58
(XNOR_STAGE_CYO59)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_59
(XNOR_STAGE_CYO60)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_60
(XNOR_STAGE_CYO61)
    MUXCY:CI->O          33   0.524   0.934  GE_STAGECY_RN_0 (GE_STAGE_CYO1)
    LUT3:I0->O            2   0.313   0.588  MAX_FIND/_N0005<0>1
(MAX_FIND/_N0005<0>)
    LUT2:I0->O            1   0.313   0.000  XNOR_STAGELUT62 (N68)
    MUXCY:S->O            1   0.377   0.000  XNOR_STAGECY_RN_61
(XNOR_STAGE_CYO62)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_62
(XNOR_STAGE_CYO63)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_63
(XNOR_STAGE_CYO64)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_64
(XNOR_STAGE_CYO65)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_65
(XNOR_STAGE_CYO66)
```

```
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_66
(XNOR_STAGE_CYO67)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_67
(XNOR_STAGE_CYO68)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_68
(XNOR_STAGE_CYO69)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_69
(XNOR_STAGE_CYO70)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_70
(XNOR_STAGE_CYO71)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_71
(XNOR_STAGE_CYO72)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_72
(XNOR_STAGE_CYO73)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_73
(XNOR_STAGE_CYO74)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_74
(XNOR_STAGE_CYO75)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_75
(XNOR_STAGE_CYO76)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_76
(XNOR_STAGE_CYO77)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_77
(XNOR_STAGE_CYO78)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_78
(XNOR_STAGE_CYO79)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_79
(XNOR_STAGE_CYO80)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_80
(XNOR_STAGE_CYO81)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_81
(XNOR_STAGE_CYO82)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_82
(XNOR_STAGE_CYO83)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_83
(XNOR_STAGE_CYO84)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_84
(XNOR_STAGE_CYO85)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_85
(XNOR_STAGE_CYO86)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_86
(XNOR_STAGE_CYO87)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_87
(XNOR_STAGE_CYO88)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_88
(XNOR_STAGE_CYO89)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_89
(XNOR_STAGE_CYO90)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_90
(XNOR_STAGE_CYO91)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_91
(XNOR_STAGE_CYO92)
    MUXCY:CI->O          33   0.524   0.934  GE_STAGECY_RN_1 (GE_STAGE_CYO2)
    LUT3:I0->O            2   0.313   0.588  MAX_FIND/_N0007<0>1
(MAX_FIND/_N0007<0>)
    LUT2:I0->O            1   0.313   0.000  XNOR_STAGELUT93 (N101)
```

```
    MUXCY:S->O          1   0.377   0.000  XNOR_STAGECY_RN_92
(XNOR_STAGE_CYO93)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_93
(XNOR_STAGE_CYO94)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_94
(XNOR_STAGE_CYO95)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_95
(XNOR_STAGE_CYO96)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_96
(XNOR_STAGE_CYO97)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_97
(XNOR_STAGE_CYO98)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_98
(XNOR_STAGE_CYO99)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_99
(XNOR_STAGE_CYO100)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_100
(XNOR_STAGE_CYO101)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_101
(XNOR_STAGE_CYO102)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_102
(XNOR_STAGE_CYO103)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_103
(XNOR_STAGE_CYO104)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_104
(XNOR_STAGE_CYO105)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_105
(XNOR_STAGE_CYO106)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_106
(XNOR_STAGE_CYO107)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_107
(XNOR_STAGE_CYO108)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_108
(XNOR_STAGE_CYO109)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_109
(XNOR_STAGE_CYO110)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_110
(XNOR_STAGE_CYO111)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_111
(XNOR_STAGE_CYO112)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_112
(XNOR_STAGE_CYO113)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_113
(XNOR_STAGE_CYO114)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_114
(XNOR_STAGE_CYO115)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_115
(XNOR_STAGE_CYO116)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_116
(XNOR_STAGE_CYO117)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_117
(XNOR_STAGE_CYO118)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_118
(XNOR_STAGE_CYO119)
    MUXCY:CI->O         1   0.042   0.000  XNOR_STAGECY_RN_119
(XNOR_STAGE_CYO120)
```

```
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_120
(XNOR_STAGE_CYO121)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_121
(XNOR_STAGE_CYO122)
    MUXCY:CI->O           1   0.042   0.000  XNOR_STAGECY_RN_122
(XNOR_STAGE_CYO123)
    MUXCY:CI->O          32   0.524   0.933  GE_STAGECY_RN_2 (GE_STAGE_CYO3)
    LUT3:I0->O            4   0.313   0.629  MAX_FIND/_N0008<0>1
(MAX_FIND/_N0008<0>)
    LUT4:I0->O            1   0.313   0.000  EQ_STAGELUT16 (N150)
    MUXCY:S->O            1   0.377   0.000  EQ_STAGECY_RN_15 (EQ_STAGE_CYO15)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_16 (EQ_STAGE_CYO16)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_17 (EQ_STAGE_CYO17)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_18 (EQ_STAGE_CYO18)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_19 (EQ_STAGE_CYO19)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_20 (EQ_STAGE_CYO20)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_21 (EQ_STAGE_CYO21)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_22 (EQ_STAGE_CYO22)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_23 (EQ_STAGE_CYO23)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_24 (EQ_STAGE_CYO24)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_25 (EQ_STAGE_CYO25)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_26 (EQ_STAGE_CYO26)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_27 (EQ_STAGE_CYO27)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_28 (EQ_STAGE_CYO28)
    MUXCY:CI->O           1   0.042   0.000  EQ_STAGECY_RN_29 (EQ_STAGE_CYO29)
    MUXCY:CI->O           3   0.524   0.610  EQ_STAGECY_RN_30 (MAX_FIND/_N0019)
    LUT4:I0->O            1   0.313   0.390  MAX_FIND/_N0013<1>1
(RECOG_NO_1_OBUF)
    OBUF:I->O                 2.851         RECOG_NO_1_OBUF (RECOG_NO<1>)
   ----------------------------------------
   TOTAL                    24.162NS (16.424NS LOGIC, 7.739NS ROUTE)
                                 (68.0% LOGIC, 32.0% ROUTE)

=========================================================================
==
CPU : 461.42 / 463.03 S | ELAPSED : 461.00 / 463.00 S

-->

TOTAL MEMORY USAGE IS 247028 KILOBYTES

NUMBER OF ERRORS   :    0 (   0 FILTERED)
NUMBER OF WARNINGS :   12 (   0 FILTERED)
NUMBER OF INFOS    :   16 (   0 FILTERED)
```