A VITERBI SLAVE FOR A LARGE-VOCABULARY REAL-TIME SPEECH RECOGNITION SYSTEM

by

Shankar Narayanaswamy

Memorandum No. UCB/ERL M89/134

14 December 1989

A VITERBI SLAVE FOR A LARGE-VOCABULARY REAL-TIME SPEECH RECOGNITION SYSTEM

by

Shankar Narayanaswamy

Memorandum No. UCB/ERL M89/134

14 December 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A VITERBI SLAVE FOR A LARGE-VOCABULARY REAL-TIME SPEECH RECOGNITION SYSTEM

by

Shankar Narayanaswamy

Memorandum No. UCB/ERL M89/134

14 December 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Contents

Chapter 1

Introduction

This report describes the design of the backtrace processor in the wordprocessing subsystem of a real-time large-vocabulary continuous-speech recognition system [1] [2] [3]. This system recognizes a 3000-word vocabulary in real time by processing 50,000 states per frame, based on a frame width of 10ms. The system has the capacity to handle 8,000 words in "off real-time".

A brief introduction to hidden Markov models and the Viterbi algorithm and a high-level explanation of the implementation of this algorithm in the system are given.

An overview of the design and operation of the wordprocessing subsystem is given followed by the design and operation of the backtrace processor chip, one of the two custom chips in the wordprocessing subsystem. The layout was created using the LagerIV Silicon Compiler [4]. The chip was fabricated in a $2\mu m$ nwell CMOS process and has 102 signal, 4 substrate, 13 power and 13 ground pins. Its die area is $6856\mu m$ by $7556\mu m$ and it has 12,760 transistors.

This is followed by a discussion of a dual-port RAM that is used in the custom chips of the wordprocessing subsystem. This RAM was designed by making extensive modifications to an existing self-timed RAM.

The chip layout of the backtrace processor is included in an appendix.

Chapter 2

Overview of the Speech Recognition System

Earlier hardware systems used dynamic time-warp algorithms [5] [6]. More recently, Bisiani et al. [7] have built a multiprocessor system that uses the hidden Markov model to recognize a 1,000 word vocabulary in 1.3 times real time. The hidden Markov model has been shown to perform better than time-warp models.

This speech recognition system uses the hidden Markov model [8] and the Viterbi algorithm [9] to achieve accurate recognition. It recognizes a 3,000 word vocabulary in real time.

2.1 The Hidden Markov Model

The hidden Markov model models statistical processes with hidden states but observeable outcomes that depend on these states. For example, consider the multi-state system shown in Figure 2.1. The arrows indicate allowed state transitions. At each timestep. a possibly unfair die is tossed; each state may have its own die. The result of this toss determines the state to which the system changes. In the figure, X is the result of the die toss. The mapping from dice toss result to next state can vary from state to state and defines the transition probability. The observer does not have access to the state sequence; all he sees is an output that is statistically dependent on the result of the die toss. An output probability is thus defined as the probability that the die toss result causes a particular output. This is called a second statistical process.

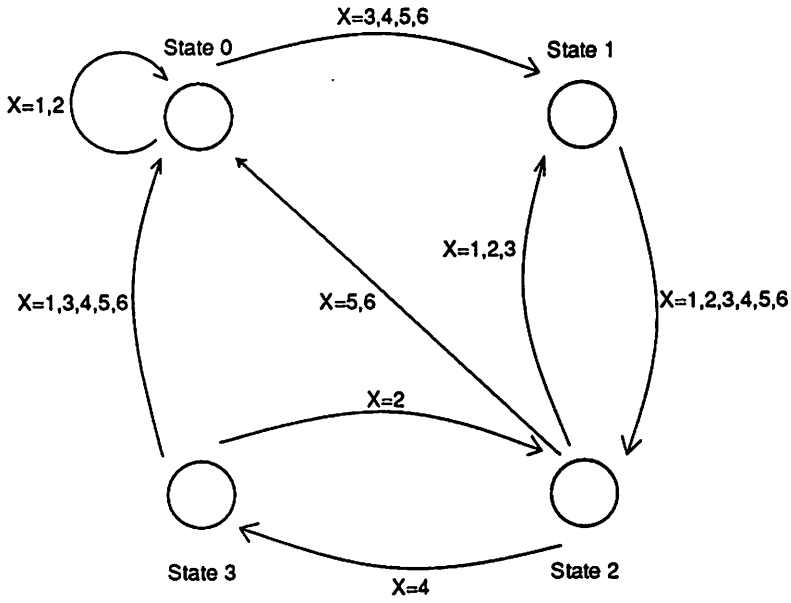

Figure 2.1: Hidden Markov model for a multi-state system

The hidden Markov model, as illustrated in the above example, is characterized by a hidden state sequence that is statistically dependent on a random process which is not observeable; we can only observe a value that is statistically dependent on the random process. This model is useful because the Viterbi algorithm can be used to efficiently determine the state sequence from the observed output process.

2.2 The Viterbi Algorithm

The Viterbi algorithm is an efficient implementation of the optimal state-sequence detector for hidden Markov models. It is best described by an example.

Consider a hidden Markov model that uses a random process to determine state transitions, as shown in the trellis in Figure 2.2. If we allow the process to start at any state and change to any state (including itself) with an associated transition probability (which can be zero), we can calculate the probability of any state sequence. At time $t-1$, each state has a probability associated with it reflecting the likelihood of being in that state. The probability of being in any state at time t can be calculated using the formula :

$$P_t(j) = \max_i [P_{t-1}(i)a_{ij}] \times b_j(O_t)$$

where $P_t(j)$ is the probability of being in state j at time t, a_{ij} is the transition probability

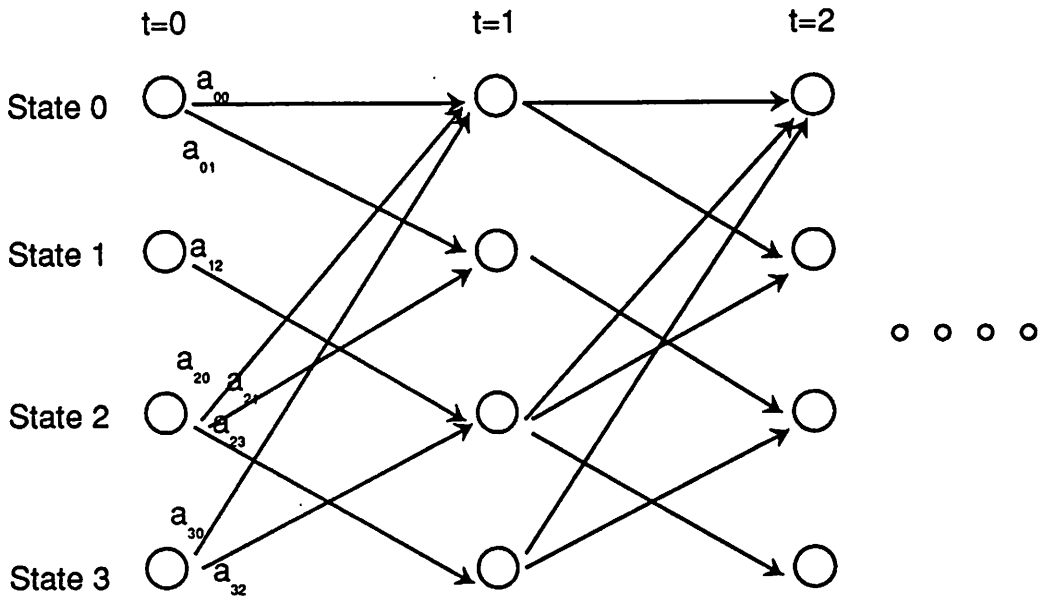

Figure 2.2: Trellis for evaluation of a hidden Markov model using the Viterbi algorithm

from state i to state j, $b_j(O_t)$ is the probability that the output of state j matches the observed output at time t, and i spans all predecessor states. This means that only the most likely path to state j is retained; all other paths are discarded. If the most probable path leading to an ending state is stored, we will then be able to trace back through the trellis to determine the most probable state sequence.

2.3 Implementation of the Viterbi Algorithm

The speech recognition system models speech with a hidden Markov model. This modeling is done on two levels. In the word-level processing subsystem, each word consists of several phonemes, and each phoneme is modeled using three states. Therefore a word is represented by several states (15 states on average) and there is a probability associated with each possible transition between states within a word. Every frame of 10 ms, the subsystem uses the Viterbi algorithm to determine the probability of being in each state of every word in the vocabulary. It uses up to four speech features as observeables (matching them against the features expected for the current state) as well as state transition probabilities to make this determination. These features may be the cepstrum, mel-cepstrum, normalized power, or other such parameters of the speech sample.

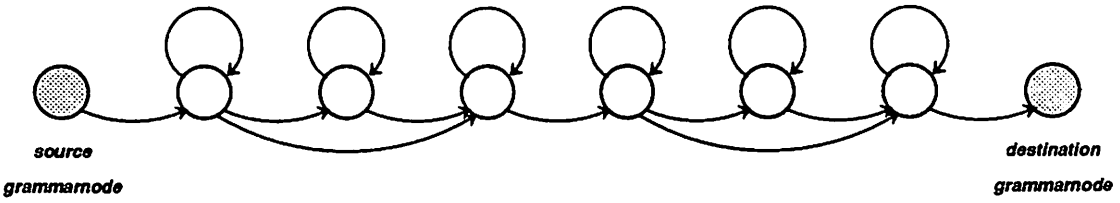

Figure 2.3: State Transition Diagram for HMM States Within a Word

In addition to states corresponding to phonemes, each word has two artificial states: a source grammarnode and a destination grammarnode. The former is the word's first state and the latter is the word's last state. They facilitate communication between the two subsystems as explained below. As shown in Figure 2.3, each word is processed left-to-right, and transitions are allowed only back to the current state and to succeeding states. Each state may be succeeded by more than one state and preceded by more than one state, but all successor states must be to the right of the current state in the state transition diagram. The exceptions are the source grammarnode, which has no predecessors but may have more than one successor, and the destination grammarnode, which may have more than one predecessor but no successors.

Since the Viterbi algorithm only requires knowledge of probabilities at time $t-1$ in order to calculate probilities for time t, it is not necessary to store the entire trellis. Therefore, the state probabilities at time $t-1$ are stored in one memory (memory 0) while the probabilities at time t are computed and stored in another memory (memory 1). During the next frame memory 1 is used to calculate probabilities at time $t+1$, which in turn are stored in memory 0. These memories are called the state probability memories.

In order to enable tracing back through the trellis at the end of the sentence, each state has a tag associated with it that points to the previous word in the trellis. This tag is stored together with the state probability and is copied between the state probability memories as needed. When the probability of the destination grammarnode is high enough, the tag and state ID are stored in a backtrace memory. At the end of the sentence, this memory is read and the linked list of tags is traversed from the last state of the most probable state sequence to its beginning, thus determining the sentence spoken.

In the grammar-level processing subsystem, each word is treated as a state. Every

frame, the subsystem tries to determine which words are ending based on destination grammarnode probabilities obtained from the wordprocessing subsystem. If a word has ended, the subsystem uses a statistical language model to determine which words can follow the word that just ended. It then updates the probabilities of the source grammarnodes of the successor words and passes these new probabilities to the wordprocessing subsystem. The system can therefore be modified to use any language model by simply replacing the grammar processing subsystem with one that uses the new language model.

Chapter 3

The Wordprocessing Subsystem

The wordprocessing subsystem is implemented using two custom chips, three large memory blocks, two FIFOs and many off-the-shelf support chips. The size of the memory blocks makes it necessary to use dynamic RAMs, which typically have cycle times of 200 ns. Since one of these blocks (the Output Memory) is accessed randomly every clock cycle, the board uses a 5 MHz clock and a non-interleaved memory architecture.

The flow of data in the subsystem is shown in Figure 3.1.

3.1 Custom Chips

The custom chips are tabulated in Table 3.1 below.

Chip Name	Designer	Die Size	Transistors	Signal Pads
Viterbi Processor	Anton Stölzle	11.6× 9.8mm^2	25,000	204
Backtrace Processor	Shankar Narayanaswamy	6.9× 7.6mm^2	12,760	102

Table 3.1: Custom chips in the wordprocessing subsystem

3.1.1 Viterbi Processor

The Viterbi Processor contains the finite state machine that drives the entire subsystem. The state transition diagram is shown in Figure 3.2.

When the subsystem is reset, it enters state 0. As soon as *startframe* is asserted.

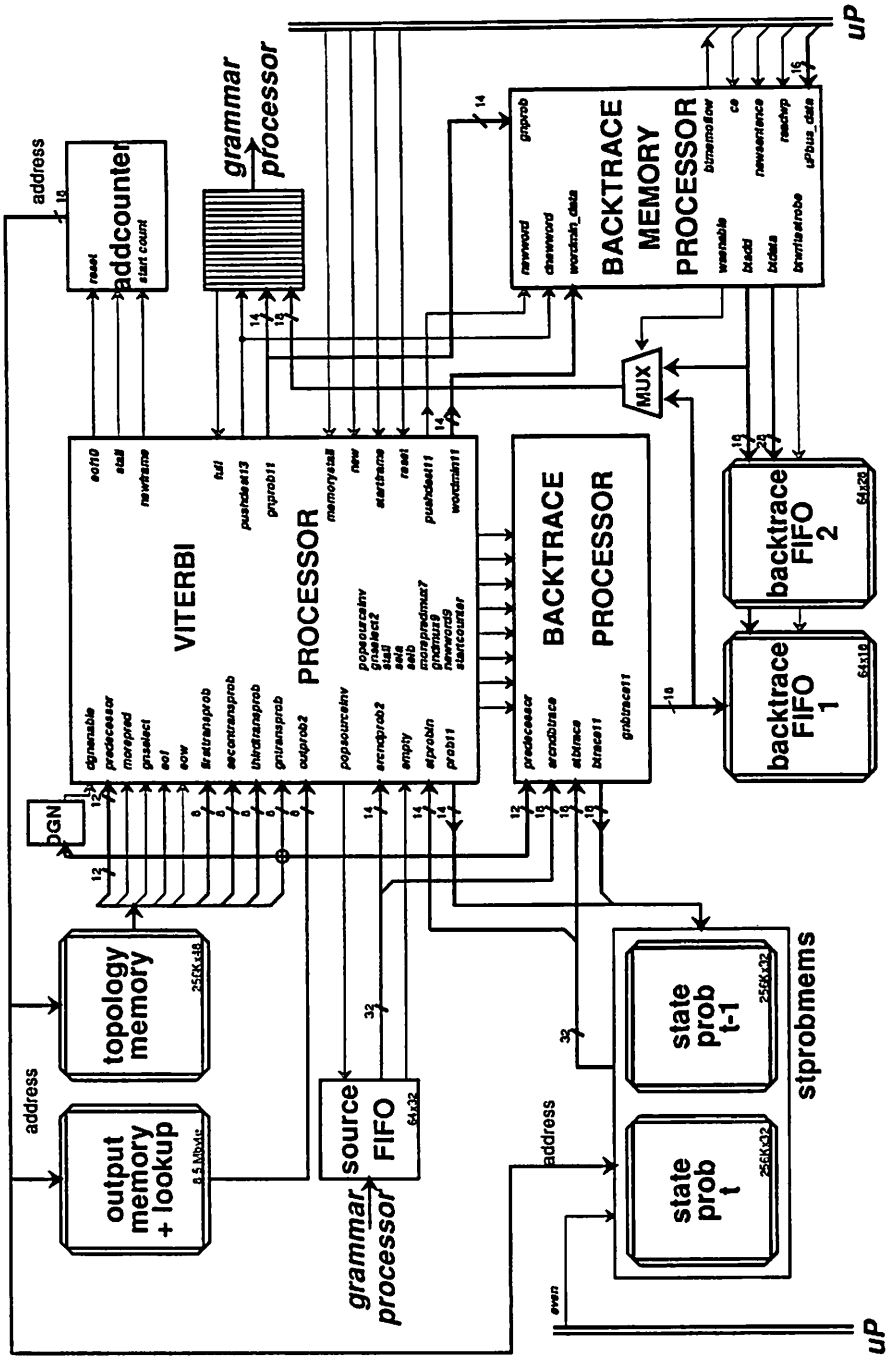

Figure 3.1: Block Diagram of the Wordprocessing Subsystem

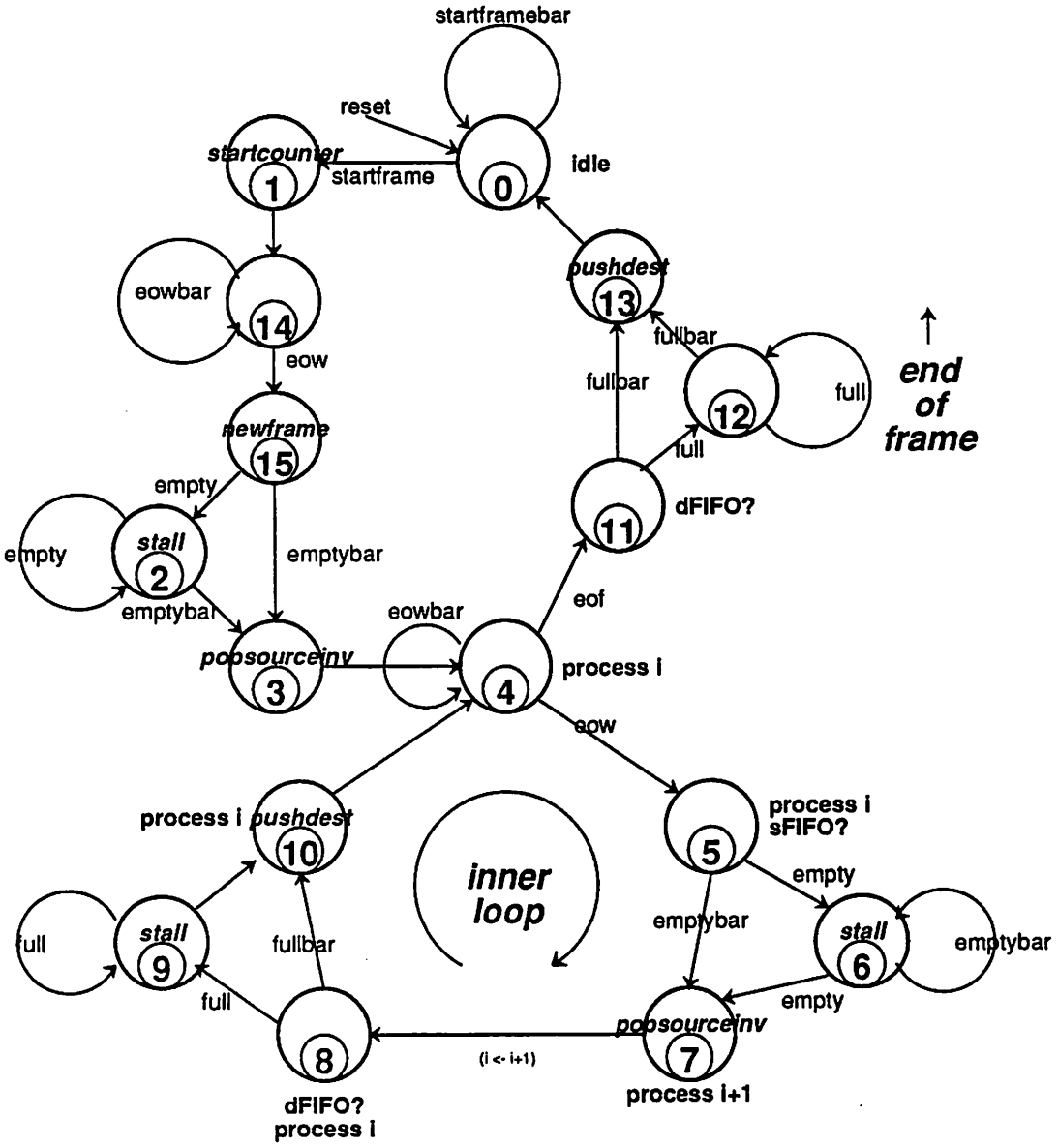

Figure 3.2: State Transition Diagram for the Finite State Machine on the Viterbi Processor

a transition to state 1 occurs and the output *startcounter* is asserted. This resets certain registers on the two custom chips. A transition to state 14 occurs on the next clock. The subsystem remains in this state until *eow* (end-of-word) goes high; this signals the processor that real data is coming on the next clock cycle. In state 15, the output *newframe* becomes active so that the counter *addcounter* begins counting up. *addcounter* holds the state address for the Output Lookup Memory, Topology Memory and State Probability Memories, which are explained in Section 3.3.

The Source FIFO is then checked to see if it has any available data. These data are the source grammarnode probabilities. If not, the subsystem enters state 2 and waits until data becomes available. *addcounter* is stalled during this state. As soon as data is available, state 3 pops the first value off the FIFO and transits to state 4, where it stays until *eow* is active again. During state 4, a word is processed. At the end of the word, the FIFO is popped by states 5, 6 and 7. In states 8, 9 and 10 the subsystem pushes the destination grammarnode probability into the Destination FIFO. This FIFO is in the grammarprocessing subsystem. The subsystem then re-enters state 4 and repeats the inner loop until *eof* (end-of-frame) goes high. It then pushes the destination grammarnode probability of the last word onto the Destination FIFO and re-enters state 0. Here it awaits the start of the next frame before repeating the entire sequence.

3.1.2 Backtrace Processor

The Backtrace Processor is the subject of Chapter 4, and a complete description can be found there.

3.2 Backtrace Memory Processor

The Backtrace Memory Processor applies a threshold to the destination grammarnode probabilities in order to determine whether the probability that the current word is ending is high enough to write it into the Backtrace FIFO. Every frame, a maximum frame probability is obtained by keeping track of the largest state probability of all the states in the vocabulary. Each frame, an offset *uPbus_data* is subtracted from the maximum frame probability of the previous frame and the result is used as the threshold. This ensures that only the most probable destination grammarnodes are written into the Backtrace FIFO.

The other function of this module is to provide the backtrace tag for the system. Whenever a word is written into the Backtrace FIFO, a new tag is generated and sent to the grammarprocessing subsystem. This tag is the backtrace ID of the entry in the Backtrace FIFO, and is generated by simply keeping count of how many words have been written into the FIFO.

The Backtrace Memory Processor is implemented using Altera PLDs.

3.3 Memory Blocks

The three memory blocks are implemented using dynamic RAMs packaged as SIMMs (Single In-line Memory Modules). There is a total of 12 Megabytes of memory in the subsystem.

3.3.1 Output Memory and Lookup

The Output Lookup Memory is a 256K x 16 lookup table for addressing the Output Memory. This allows the reduction of the size of the Output Memory by a factor of 256. It uses two 256K x 8 SIMMs and is loaded only once, during system startup.

The Output Memory stores the probability that the current speech sample matches the current state. It consists of four 2M x 8 banks. It is implemented as 8 SIMMs and is loaded only at system startup. Its address is determined by 15 bits of data read from the Output Lookup Memory and by four 8-bit of data signals obtained from speech features.

The output Memory can operate in two modes. In mode 0, only one speech feature is used. In this case, all four speech input signals are identical. Fifteen bits of data read out of the Output Lookup Memory are combined with 8 bits from the speech input to produce a 23-bit address. This address points to the location in the Output Memory that stores the desired probability.

In mode 1, each of the four 2-Megabyte banks is addressed by 13 bits from the Output Lookup Memory and 8 bits from speech features; four different speech features can therefore be handled. The four probabilities so obtained are added together and the result is sent to the Viterbi Processor.

3.3.2 Topology Memory

The Topology Memory holds the state transition probabilities. It is identical to the Output Lookup Memory except for its size, 256K x 48. Its first 8 bits (7-0) hold the probability that the current state is a predecessor to the destination grammarnode. Bits 31-24, 23-16 and 16-8 hold the transition probabilities of the three predecessors. Bits 43-40, 39-36 and 35-32 hold the relative positions of the three predecessors of the current state. Bits 44 and 45 hold end-of-word and end-of-frame. Bit 46 tells the subsystem if one of the predecessors is the source grammarnode while bit 47 tells the subsystem if there are more predecessors for the present state.

3.3.3 State Probability Memory

The state probability memory stores the state probabilities of all the states in the subsystem at time $t-1$ and time t. It has two banks of 256K x 32 each. The first 14 bits store the probabilities, while the remaining 18 bits hold the backtrace tag that points to the previous word. These are the memories mentioned in Section 2.3.

3.4 FIFOs

The two FIFOs are primarily meant for asynchronous communication between the wordprocessing subsystem and other parts of the speech recognition system.

3.4.1 Source FIFO

The Source FIFO is 64 words deep and 32 bits wide. It passes data (updated source-grammarnode probabilities) from the grammarprocessing subsystem to the wordprocessing subsystem asynchronously. There is an identical FIFO in the grammarprocessing subsystem that passes data (updated destination grammarnode probabilities) in the other direction.

3.4.2 Backtrace FIFO

The Backtrace FIFO is 64 words deep and 48 bits wide, but only 44 bits are used. It provides an asynchronous interface between the wordprocessing subsystem and the rest of the speech recognition system for passing backtrace information. When it is almost full,

it is popped onto the VME bus and the data is stored on a general-purpose processor board for further processing after the end of the sentence.

The first 18 bits are the backtrace tag, which point to the word that preceded the current state. This is obtained from the Backtrace Processor. The next 10 bits hold the word ID of the current word while the last 14 bits hold the destination grammarnode probability of the current word. This probability is kept for recordkeeping purposes only; it is not used in the speech recognition recognition algorithm. The word ID and destination grammarnode probability are obtained from the Backtrace Memory Processor.

3.5 Miscellaneous Chips

DGN (see Figure 3.1) is an 8-input AND gate that decodes the 8-bit signal *gntransprob* so that if all the bits are high, *dgnenable* goes high. When *dgnenable* goes high, the probability of a transition to the destination grammarnode is very small, so the Viterbi Processor assumes that the current state cannot transit to the destination grammarnode.

MUX is an 18-bit two-to-one multiplexer that selects the backtrace data that is passed to the grammarprocessing subsystem. If the current word is not being stored in the backtrace memory, *gnbtrace11* from the Backtrace Processor is selected, otherwise the location where the current word is being stored is selected.

Chapter 4

The Backtrace Processor

The Backtrace Processor is a slave to the Viterbi Processor, processing the backtrace tag associated with each state. Each tag is a pointer to the previous word; the tags therefore form a linked list of pointers from the end of the state sequence to the beginning. This allows the system to trace a path through the backtrace memory to determine the spoken sentence.

The Backtrace Processor consists of two main parts. The Backtrace Datapath (*btracedp*) does the actual processing of the backtrace tag. The Upper Datapath (*upperdp*) stores the backtrace tags for all the states in the word currently being processed and feeds the correct tag to the Backtrace Datapath every clock cycle. These operations are described in greater detail below. Figure 4.1 shows the interconnection between these two parts.

4.1 The Upper Datapath

The Upper Datapath consists of two parts, the Predecessor Selector (*predsel*) and the Predecessor Address Datapath (*predadd*). In the Viterbi algorithm, the probability for each state is determined from the probabilities of its predecessors. Most words have at most 15 states. Therefore, an efficient way to get the predecessor data to *btracedp* is to read the data for these 15 states sequentially onto the chip from the State Probability Memory and store it in a 16-word on-chip RAM. The data can then be fed into *btracedp*. Since the hidden Markov model may only make left-to-right transitions (Figure 2.3), the "max" function from Section 2.2 does not consider states that occur later in the word. This allows the custom chips to load the data into the on-chip RAM sequentially.

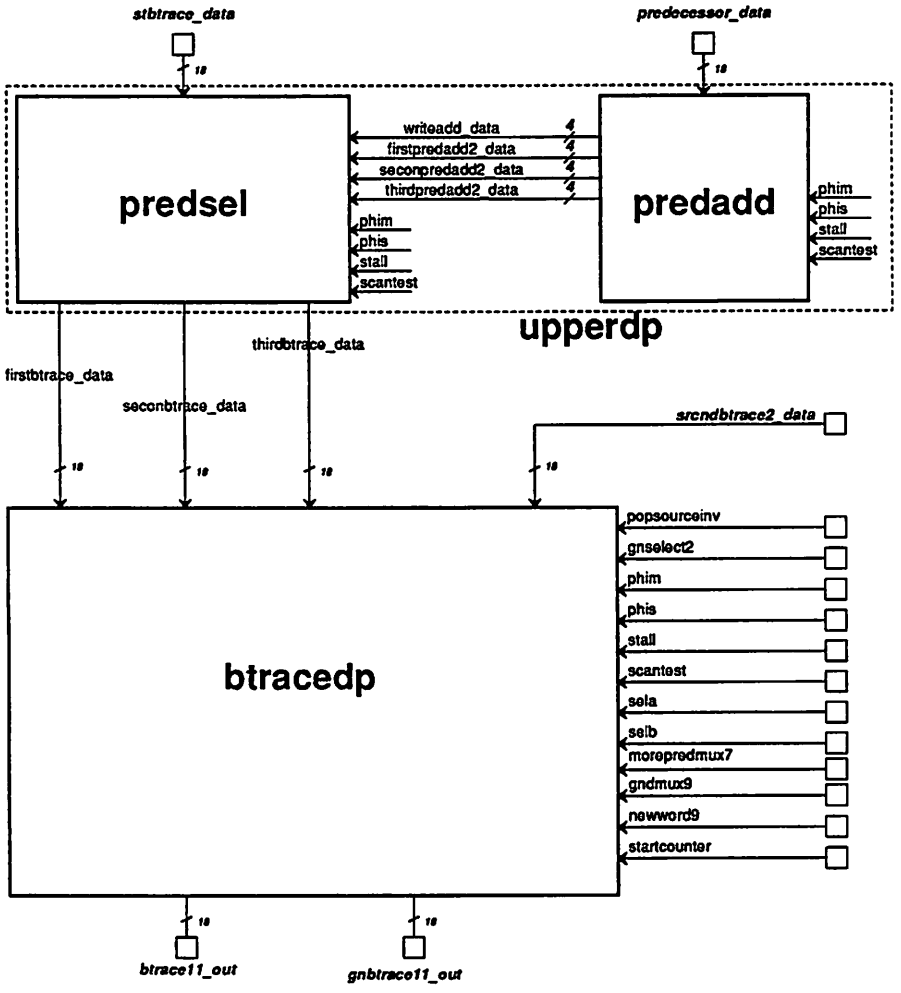

Figure 4.1: Block Diagram of the Backtrace Processor

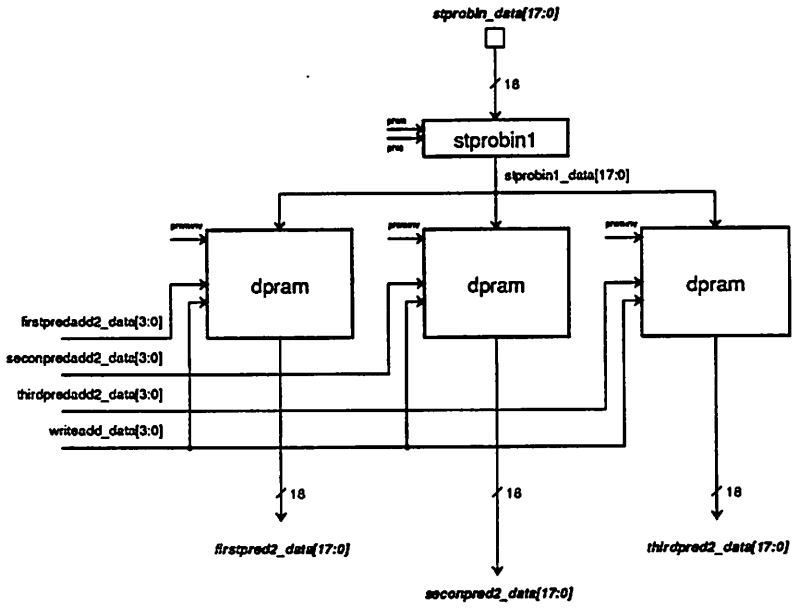

Figure 4.2: Block Diagram of the Predecessor Selector (*predsel*)

In order to parallelize the "max" function in the Viterbi algorithm, three dual-port RAMs are used. This number was chosen because most words have at most three predecessors. In case of a larger number of predecessors, the signal *morepredmux7* is active and the probabilities of the next three predecessors are read into the dual-port RAMs for processing. These RAMs comprise *predsel*. They store exactly the same data but they are read from different locations at the same time, corresponding to three different predecessors. The block diagram for *predsel* is shown in Figure 4.2.

The read and write addresses for the dual-port RAM are calculated in *predadd*. The write address (*writeadd_data*) is generated by a counter formed by *inaddcounter1* and the adder feeding *inaddmux*; the read addresses are calculated by adding offsets to the write address. These offsets are obtained from the value of the signal *predecessor_data*, which is read from the Topology Memory. The block diagram for *predadd* is shown in Figure 4.3. At the beginning of the frame, the signal *startcounter* drives *inaddmux* to reset *writeadd_data*. This is necessary for simulations only; the hardware is insensitive to the starting point of the counter since the read addresses generated by *predadd* are offset from *writeadd_data*.

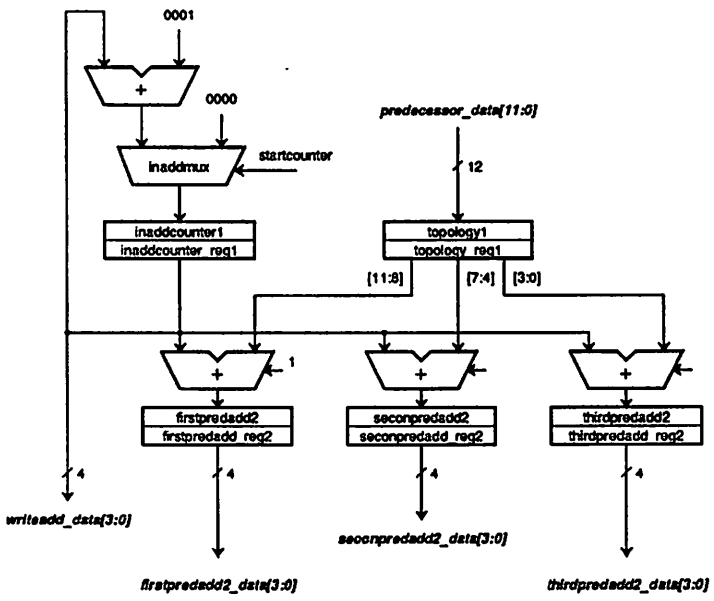

Figure 4.3: Block Diagram of the Predecessor Address Datapath (*predadd*)

4.2 The Backtrace Datapath

The Backtrace Datapath was broken up into two cascaded datapaths to obtain two square blocks rather than a single rectangular block in the chip's layout. This facilitates easier floorplanning. A block diagram for this datapath is shown in Figure 4.4.

The Backtrace Datapath is heavily pipelined. Its pipeline stages are related to the propagation delay of the logic on the Viterbi Processor that drives the multiplexer control signals. This logic is implemented in a PLA on the Viterbi Processor.

The multiplexer *MUX1* selects between the signal *srcndbtrace_data* from the source FIFO and the data stored in the register *srcndbtrace_reg2*. *srcndbtrace_data* is the tag associated with the source grammarnode and is selected at the beginning of a word. For the rest of the word, *MUX1* selects the data stored in *srcndbtrace_reg2*. This data is actually the output data from *MUX1*. All the registers on the chip are dynamic and must therefore be refreshed periodically; the above arrangement effectively converts *srcndbtrace_reg2* into a static register that is loaded at the beginning of every word.

The inputs *firstbtrace2_out*, *seconbtrace2_out* and *thirdbtrace2_out* are the data read out from the dual-port RAMs in *predsel*. They are the tags corresponding to the three predecessors of the current state. *MUX2* passes the data from *srcndbtrace_reg2* only when

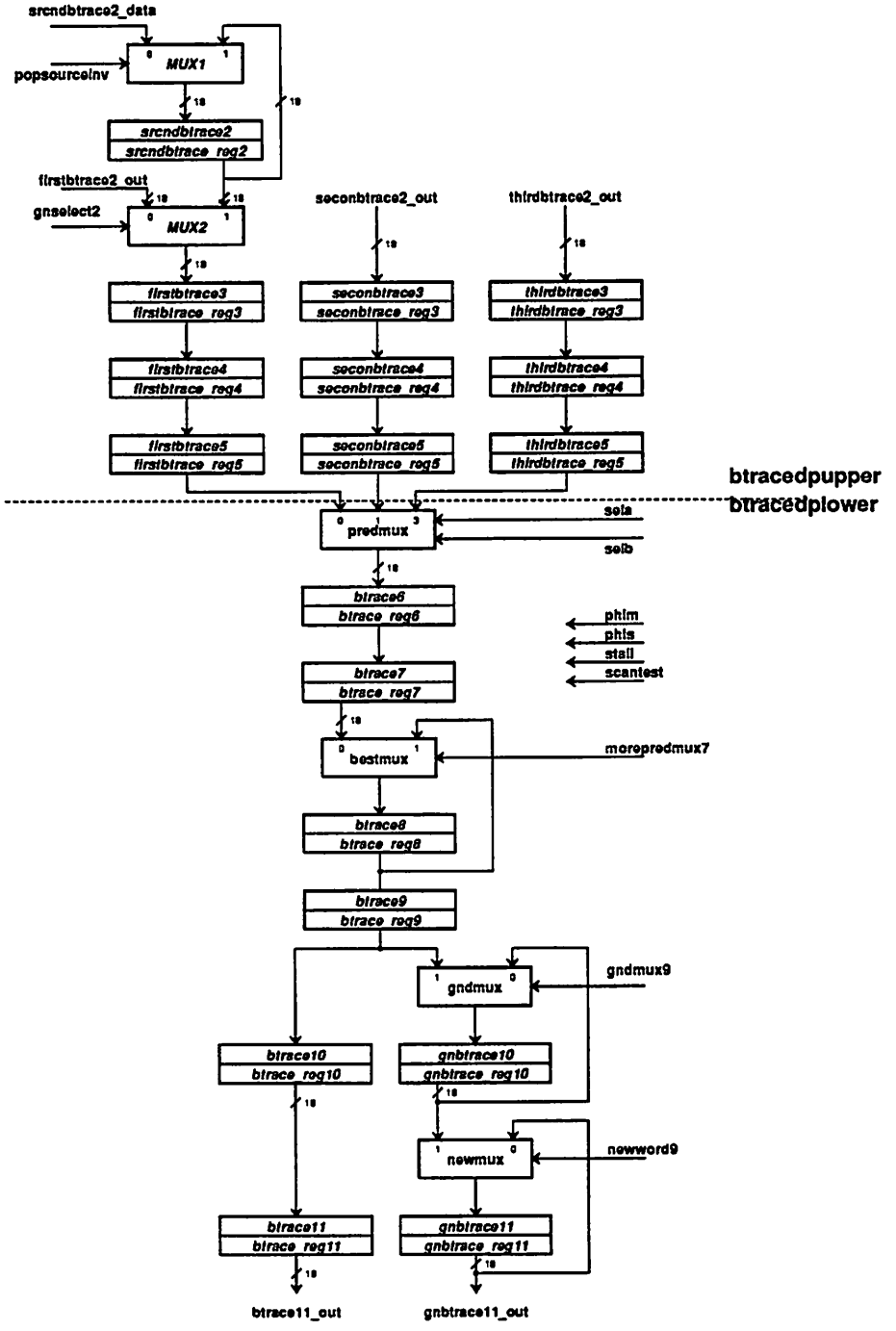

Figure 4.4: Block Diagram of the Backtrace Datapath (*btracedp*)

one of the predecessors of the current state is the source grammarnode.

The control inputs *sela* and *selb* are derived in the Viterbi Processor from the output of the "max" function in the Viterbi algorithm (Section 2.2 and they cause *predmux* to select the tag of the most likely predecessor.

The multiplexer *bestmux* uses the control input *morepredmux7* to allow retention of the tag of the previous most likely predecessor in case there are more than three predecessors. In this case, the Viterbi Processor compares the probability of the most likely of the second three predecessors to the probability of the most likely of the first three predecessors. The more probable predecessor's tag is selected by *bestmux*.

The multiplexer *gndmux* allows *gnbtrace_reg10* to store the most probable predecessor to the destination grammarnode. If the current state is more likely than any other state to be the predecessor of this grammarnode, then *gndmux* selects the output of *btrace_reg9*.

At the end of the word, *newmux* causes *gnbtrace_reg11* to store the tag of the most probable predecessor to the destination grammarnode for the duration of the next word. This allows the data to be stable for more than one clock cycle, which allows the Backtrace Memory Processor more time to push data into the Backtrace FIFO.

In total, there are eighteen 18-bit registers and seven 18-bit two-to-one multiplexers in the Backtrace Datapath..

4.3 Clocking Strategy

The figures shown thus far in this chapter do not show the clocking strategy, but it is nevertheless important. There are two outside constraints imposed on the chip that affect the clocking strategy. Firstly, the need for a testable circuit made it desirable to implement all registers using scanpath registers. The schematic for the scanpath register used in this chip is shown in Figure 4.5. It requires three clocks: the usual master and slave clocks and the shift clock for shifting data into and out of the scanpath.

Secondly, the chip must be stalled during the dynamic RAMs' refresh cycles. These constraints, coupled with the need to buffer the clock, led to the use of standard cells for gating as well as buffering the clocks. The standard cells are physically large and provide a large output current drive. The logic realized in the standard cells is shown in Figure 4.6. This figure shows the logic for driving five registers; on the chip each register is driven by one logic module.

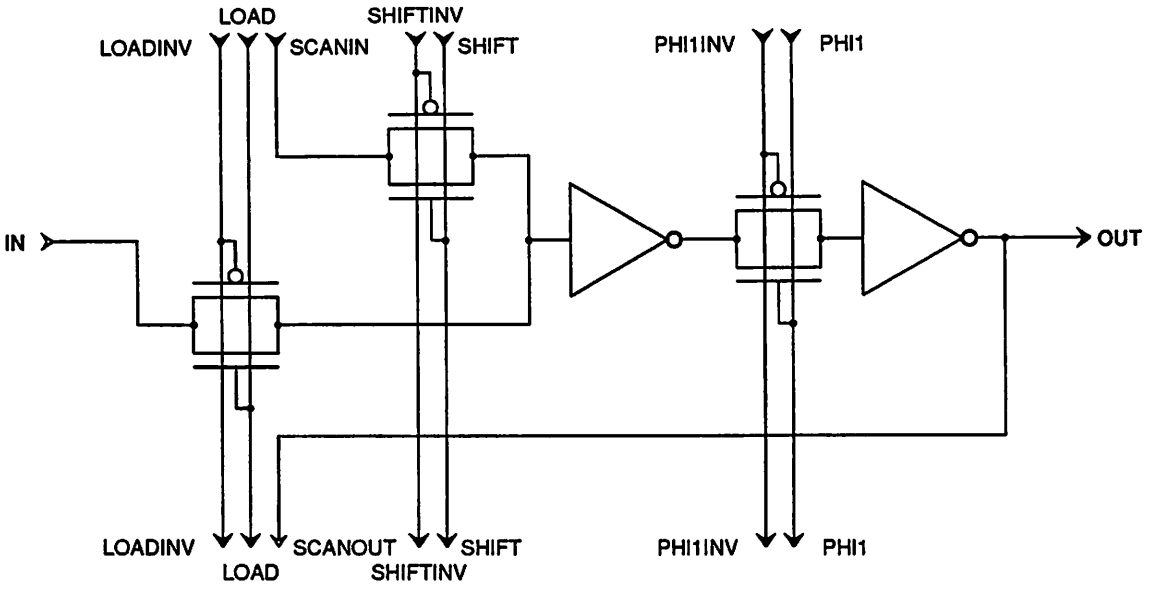

Figure 4.5: Schematic of the Scanpath Register

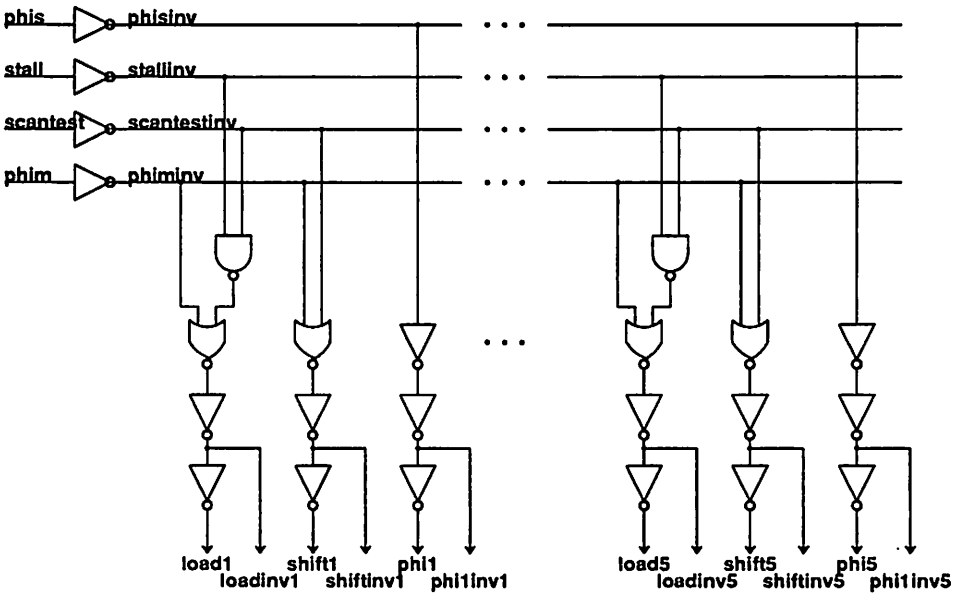

Figure 4.6: Schematic of the Scanpath Register Clock Drivers

4.4 Verification

Chip verification was done on two levels, design simulation and hardware testing.

4.4.1 Simulation

The LagerIV Silicon Compiler produces chip layout in Magic [10] format. The layout was extracted and used by RSIM to simulate the chip. RSIM models transistors as current sources and uses lumped node capacitances to determine approximate timing information. Test vectors were fed into RSIM and the results compared to output vectors obtained from THOR [11]. THOR is a functional simulator which uses a hierarchical behavioral description written in C. The entire speech recognition system was modeled using THOR even before the chips were built, and the chip design actually used the THOR description as the blueprint. The THOR modeling allowed verification of the system's functionality before building hardware and easy design verification by simulation.

4.4.2 Scanpath Testing

The Backtrace Processor is heavily pipelined, and every register on the chip is a scanpath register. This allows easy testing of the combinatorial parts of the chip. However, the only testing machine available was a Tektronix DAS 9100, which was not very convenient for the purpose. Instead, an IBM-PC based scanpath testing system was developed by Anton Stölzle, Shankar Narayanaswamy and Robert Yu. This testing system is general enough to test all chips that use the above scanpath scheme.

Testing System

The testing system uses a custom board that interfaces between the IBM-PC and the user's test board. The user specifies a test vector to be fed into the scanpath as a single line of bits in a file on the PC. These bits are loaded by software into a memory on the custom board. The custom board then downloads the bits into the chip to be tested on the test board, runs the chip for one exactly clock cycle, and reads the scanpath out again into the on-board memory. This output data is then copied into a file on the IBM-PC in exactly the same format as the input test vector. Copying data to and from the custom board is done by software written in TurboC running on the PC.

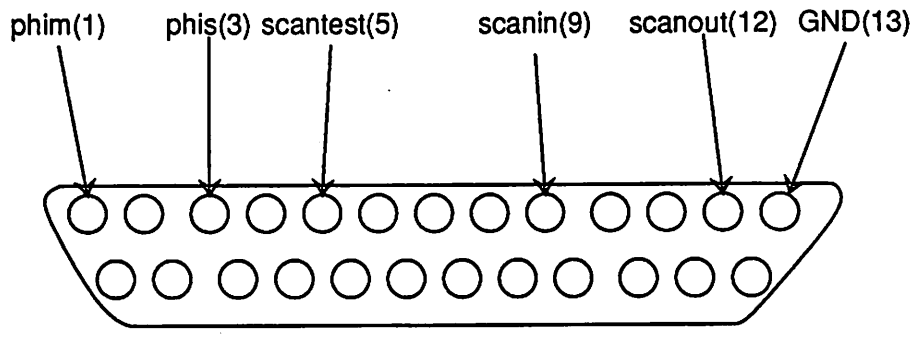

Figure 4.7: Pinouts of the ribbon cable connector for the scanpath testing system

The custom board communicates with the test board by means of a 5-signal protocol. *phim* is the master clock, *phis* is the slave clock, *scanin* is the data to be loaded into the scanpath and *scanout* is the data read out of the scanpath. *scantest* is a control signal which tells the test board that the scanpath is active; when it goes low (for one clock cycle) the test board knows that the scanpath is not being accessed and that it may run the chip. There is also a ground reference pin. The pinouts on a 25-pin ribbon cable connector are shown in Figure 4.7.

Test Board

The Backtrace Processor is packaged in a 132-pin Pin-Grid Array (PGA). Its pins are therefore not easily accessible for connection of pattern generators and data acquisition probes. The test board holds the chip in a ZIF socket and brings out all its pins to two rows of wire-wrap headers. Power and ground rail headers encircle the chip. There is also a boundary-scan shift register on the board to allow control of the chip's control inputs. This 8-bit register is connected to the beginning of the scanchain. Finally, two adjacent rows of 13-pin headers are placed at the edge of the board for connecting the ribbon cable to the scanpath testing system.

Format Conversion Software

Since the data format required by the testing system is not very readable, some software was needed to translate data to and from a friendlier format. This program, called *scanpath*, accepts an input file that has four columns per line and one line per scanpath register in the scanchain. From this, it generates the correct sequence of bits for the testing system.

The lines are arranged in order of closeness of the register to the beginning of the scanchain: the closest register being on the first line of the file. The first column of each line is the name of the register. The second column is the number of bits in the register. The third column contains a single character: "L" if the least significant bit of the register is closest to the beginning of the scanchain and "M" otherwise. This allows the user to specify data as a regular (rather than bit-reversed) number and let the computer reverse it for him. Lastly, the fourth column contains a hexadecimal number representing the value to be loaded into the register. If this number has too many bits, the most significant bits are truncated until the correct number of bits is obtained. This allows unambiguous loading of registers that have widths that are not multiples of four bits.

Below is an example of the input file for the Backtrace Processor. Note that the first register is the boundary-scan register, which is loaded as part of the scanchain to allow easy loading of test signals into the chip's control inputs.

```
Nw9Gm9Mp7SaSbGs2PsiNf 8 M FF
stprobin 18 L FFFFF
inaddcounter_reg1_1 4 M A
topology_reg1_1 4 L A
topology_reg1_2 4 M A
topology_reg1_3 4 L A
firstpredadd_reg2 4 M A
seconpredadd_reg2 4 L A
thirdpredadd_reg2 4 M A
btrace_reg6 18 M 11111
btrace_reg7 18 L 22222
btrace_reg8 18 M 33333
btrace_reg9 18 L 44444
gnbtrace_reg10 18 M 55555
gnbtrace_reg11 18 L 66666
btrace_reg10 18 M 77777
btrace_reg11 18 L 88888
srcndbtrace_reg2 18 M 99999
firstbtrace_reg3 18 L AAAAA
```

```
firstbtrace_reg4 18 M BBBBB
firstbtrace_reg5 18 L CCCCC
seconbtrace_reg3 18 M DDDDD
seconbtrace_reg4 18 L EEEEE
seconbtrace_reg5 18 M FFFFF
thirdbtrace_reg3 18 L 00000
thirdbtrace_reg4 18 M 11111
thirdbtrace_reg5 18 L 22222
```

The string of bits resulting from processing the above input file is used by the testing system, which in turn produces a similar output string of bits. This output vector may then be compared to the input vector using *scanpath*. The program will produce a file that has the same format and content as the earlier four-column file but with a fifth column that holds (in hexadecimal and, if necessary, bit-reversed) the value obtained from the testing system.

Below is an example of the output file for the Backtrace Processor. Note that the contents of the boundary-scan register are unchanged.

```
Nw9Gm9Mp7SaSbGs2PsiNf 8 M FF FF
stprobin 18 L FFFFF 00000
inaddcounter_reg1_1 4 M A 0
topology_reg1_1 4 L A 0
topology_reg1_2 4 M A 0
topology_reg1_3 4 L A 0
firstpredadd_reg2 4 M A 4
seconpredadd_reg2 4 L A 4
thirdpredadd_reg2 4 M A 4
btrace_reg6 18 M 11111 22222
btrace_reg7 18 L 22222 11111
btrace_reg8 18 M 33333 33333
btrace_reg9 18 L 44444 33333
gnbtrace_reg10 18 M 55555 04444
gnbtrace_reg11 18 L 66666 15555
btrace_reg10 18 M 77777 04444
```

```
btrace_reg11 18 L 88888 37777
srcndbtrace_reg2 18 M 99999 19999
firstbtrace_reg3 18 L AAAAA 19999
firstbtrace_reg4 18 M BBBBB 2AAAA
firstbtrace_reg5 18 L CCCCC 3BBBB
seconbtrace_reg3 18 M DDDDD 3FFFF
seconbtrace_reg4 18 L EEEEE 1DDDD
seconbtrace_reg5 18 M FFFFF 2EEEE
thirdbtrace_reg3 18 L 00000 3FFFF
thirdbtrace_reg4 18 M 11111 00000
thirdbtrace_reg5 18 L 22222 11111
```

Results

Of the 30 chips fabricated, 18 passed the scanpath test based on 6 different test vectors.

Chapter 5

The Dual-Port RAM

In the subcell *predsel* on the Backtrace Processor (Figure 4.2), the predecessor tags are stored in a dual-port RAM so that these tags can be available for processing while being updated. The Viterbi Processor also uses *predsel* and therefore the dual-port RAM. Since the LagerIV cell library did not have a dual-port RAM, it was necessary to design one from scratch.

5.1 Specifications

The dual-port RAM had to meet several specifications:

- It must allow independent reading and writing.
- It must have separate input and output data busses.
- It must not require clocks.
- It must operate at 5 MHz for a 16 word $\times$ 32 bit RAM.
- It should occupy minimum area.
- It may be dynamic since the processors that use it will write the entire memory sequentially, thereby refreshing it.
- If the read address and write address are the same, then the data read must be the data currently being written in.

- It must be assembled by the tiling generator TimLager in the LagerIV Silicon Assembly System.

At the time, a self-timed RAM designed by Brian Richards was available. Since it would take longer to design a RAM from scratch, this RAM was modified to suit the above specifications.

5.2 I/O Description

The dual-port RAM has an input bus (*in*), an output bus (*out*), a read address bus (*read_addr*), a write address bus (*write_addr*), a write enable (*write*) and a writestrobe ($\overline{pre}$). If *in* and *out* are N bits wide, then there are $log_2 N$ words in the RAM.

write is internally ANDed with $\overline{pre}$. When the user writes data into the RAM, he should enable *write* and, while *write_addr* and *in* are valid, strobe $\overline{pre}$ low. When reading data, simply set *read_addr* to the correct address and, after some propagation delay, *out* will carry valid data.

The read and write operations are completely independent. If a particular address is being written and read at the same time, *out* will initially show the old data but, after some propagation delay, it will show the new data. This delay has been verified to be less than 200 ns, as explained in Section 5.4 below.

5.3 Block-Level Description

A schematic diagram of the dual-port RAM is shown in Figure 5.1. The input data is buffered and then fed into a standard 3-transistor dynamic RAM cell. The data stored in this cell is detected with the aid of a pseudo-NMOS column precharge transistor, and is then buffered on its way out of the RAM. Address decoding is done by tiling together the appropriate cells.

5.4 Verification

Verification was done on several levels. First, the pseudo-NMOS column pull-up was simulated with SPICE [12]. Then the complete RAM was simulated using RSIM. Lastly, a fabricated chip containing the dual-port RAM was tested on the Tektronix DAS 9100.

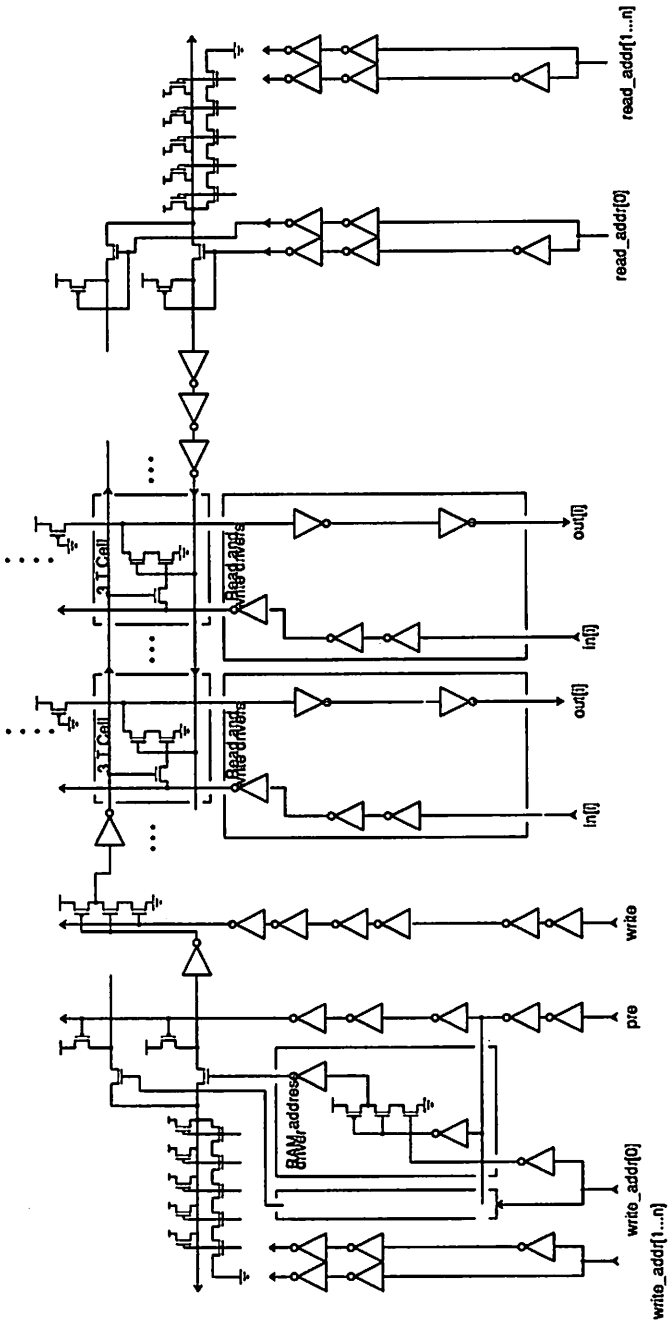

Figure 5.1: Schematic Diagram of the Dual-Port RAM

5.4.1 SPICE Simulations

One major difference between the dual-port RAM and the self-timed RAM is in the column pull-up. The former uses a pseudo-NMOS column pull-up whereas the latter uses a dynamic pre-charge. This modification meant that the size of the pull-up transistor had to be decreased to avoid it overwhelming the pull-down transistor in the 3-T RAM cell. However, it cannot not be too small; otherwise it cannot pull the column voltage up fast enough if a "1" is read after a "0".

The final transistor size was chosen to be 4λ long by 3λ wide, where λ is a scale factor that depends on the technology. For example, λ is set to 1.0 in a $2.0\mu m$ technology. This means that the pull-up transistor was $4\mu m$ by $3\mu m$ in a $2\mu m$ technology. SPICE simulations showed the column rise time to be 6 ns and the column voltage swing to be from 5 Volts to 0.38 Volts, with the transition in output voltage levels occurring at a DC input voltage of 1.5 Volts at the gate of the storage transistor. This is clearly sufficient for the purpose. The SPICE deck was created automatically by ext2spice [13] from the layout itself, so linear and nonlinear capacitances were simulated accurately. Resistive effects in the column were not modeled in the SPICE simulations, but they would merely slow down the RAM and it was fast enough that resistive effects would not render the RAM unusable.

5.4.2 RSIM Simulations

In order to test the functionality of the dual-port RAM, the layout was extracted and an input file for RSIM was generated from this layout. RSIM models node capacitances and the current drive of transistors to give an approximate timing analysis of the circuit. This is also useful for making sure that the chip runs fast enough.

Simulations showed that the RAM was functional at 10 MHz.

5.4.3 Chip Testing

A 16 word $\times$ 32 bit version of the dual-port RAM was fabricated in a $1.6\mu m$ n-well process by MOSIS and tested on a Tektronix DAS 9100. The testing strategy was to write every memory location with data, and then to read the data sequentially over and over again until the DAS ran out of storage space (about 500 reads). This allowed verification that the data in the RAM remained valid for a reasonable length of time. Since reading

and writing are independent, the test also included reading and writing the same location simultaneously.

Of the 30 chips tested, 24 were functional at a supply voltage of 5.25 Volts. None worked off a 5-Volt supply. The highest clock rate used was 5 MHz, as the DAS could not meet some of the timing constraints at higher clock speeds. This is sufficient since the Speech Recognition System operates at 5 MHz.

The dual-port was verified, however, when the Backtrace Processor was tested. Data could be written and read reliably from the dual-port RAMs on the Backtrace Processor (which was fabricated in a $2\mu m$ technology) at a supply voltage of 5 Volts.

Chapter 6

Conclusions

The backtrace processor for the wordprocessing subsystem of a large vocabulary real time speech recognition system has been designed, fabricated and verified. It uses an on-chip dual-port RAM that was designed by modifying an existing self-timed RAM. The chip was assembled, laid out and routed using the LagerIV Silicon Compiler System. It was simulated using THOR and RSIM.

Improvements may be made to the wordprocessing subsystem to increase its capabilities. The subsystem may be parallelized by using several boards that each recognizes a 3,000 word vocabulary. The output memory could be doubled in capacity by using 4-Mbyte SIMMs instead of the current 1-Mbyte ones. The backtrace memory could be transformed into a FIFO so that the backtrace information may be transferred asynchronously to the system CPU during the subsystem's operation. This reduces the subsystem's latency. Finally, a lot of the multiplexing and DRAM control could be put into a custom chip to save board area. These changes are planned for future versions.

Appendix A

Chip Layout

A plot of the Backtrace Processor is shown in Figure A.1. The major blocks can be clearly distinguished.

At the top of the plot is *btracedp*. *btracedpupper* (labelled *BDPUP*) is on the left while *btracedplower* (labelled *BDPLO*) is on the right. *predadd* is on the lower left of the chip and *predsel*, on the lower right, is clearly distinguished by its three identical dual-port RAMs.

The chip has 102 signal pads, 13 power pads and 13 ground pads. It uses 12,760 transistors and occupies $7.6mm^2$ of silicon area in a $2\mu m$ nwell process.

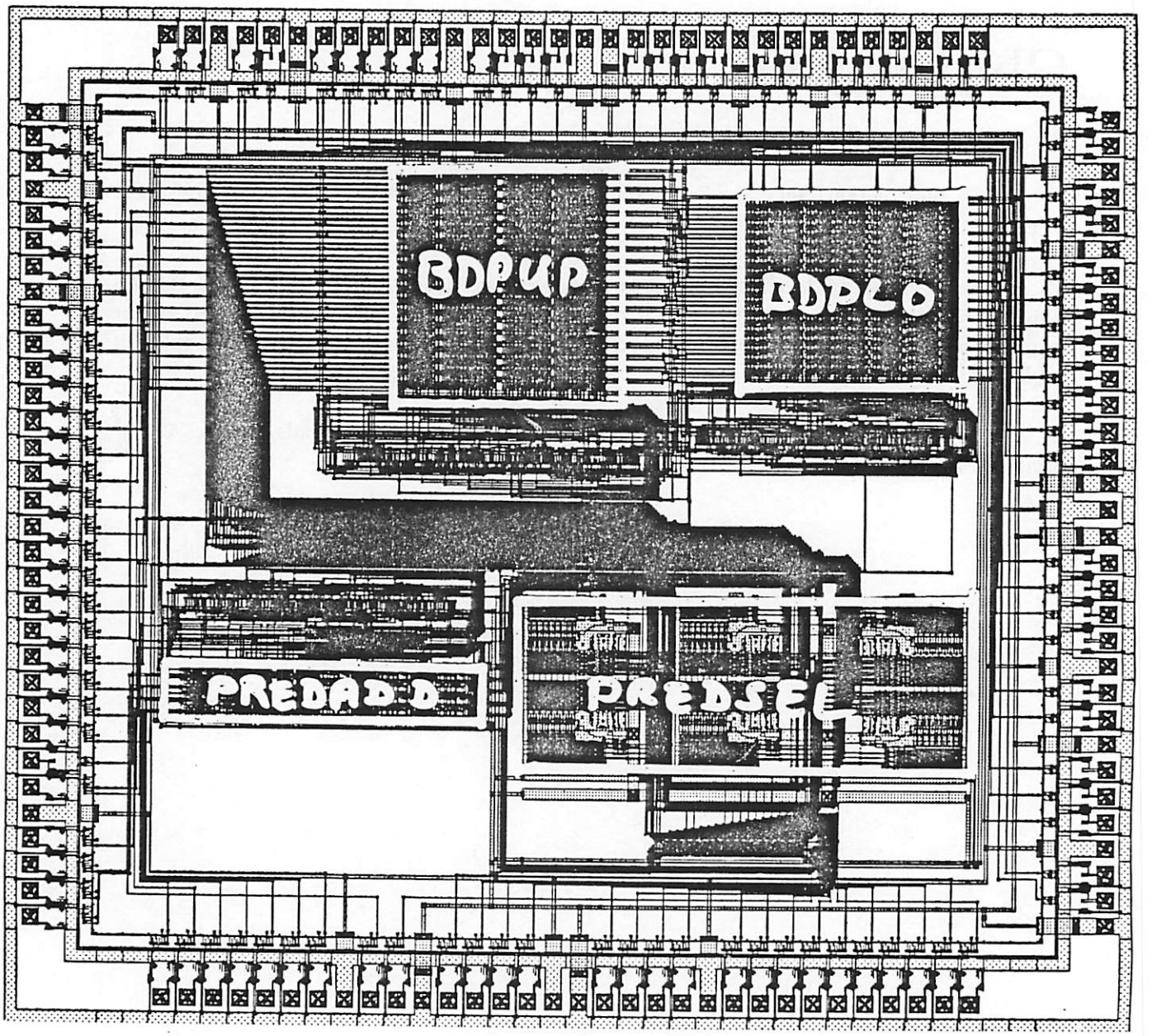

Figure A.1: Chip Photo of the Backtrace Processor

Appendix B

Chip Pinouts

The pinouts of the Backtrace Processor are shown in Figure B.1. The package has 102 signal pins, 13 power pins, 13 ground pins and 4 substrate pins in a 132-pin Pin-Grid Array (PGA).

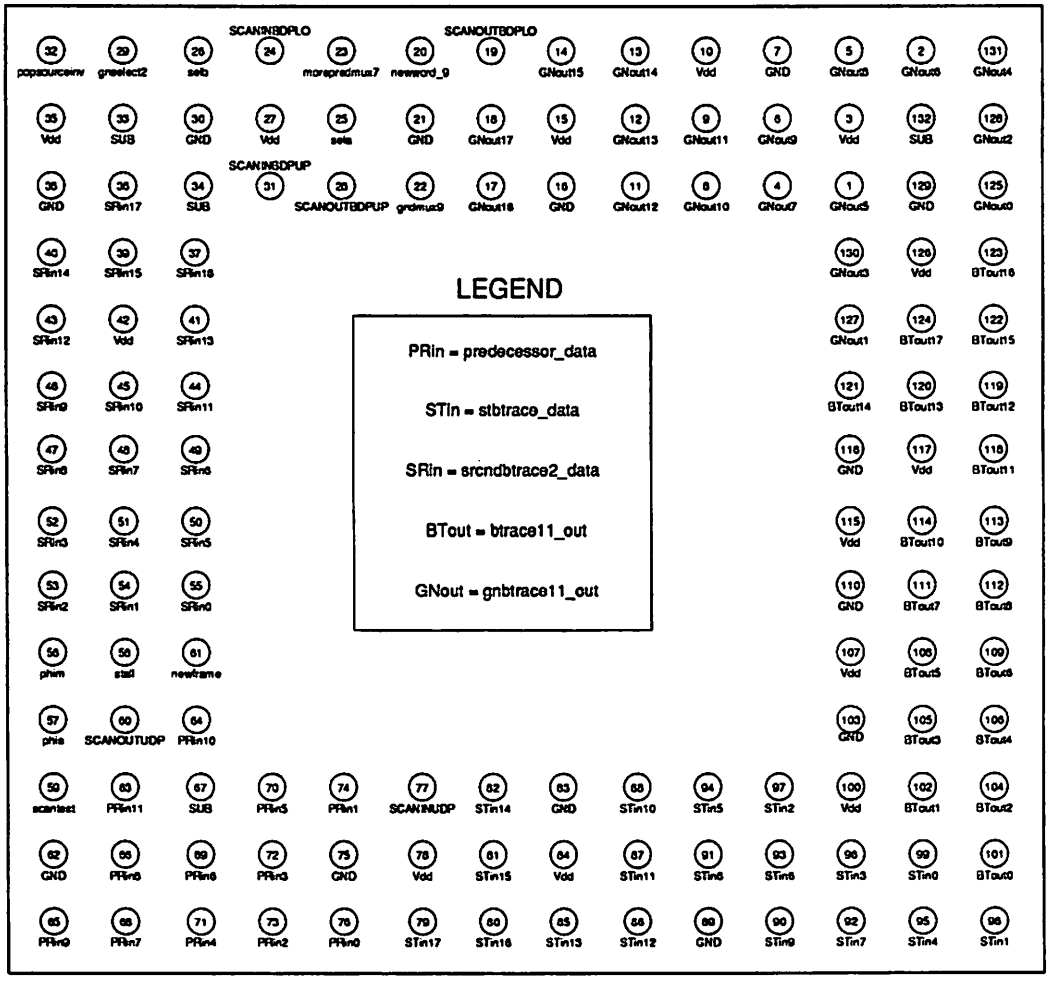

Figure B.1: Pinouts of the Backtrace Processor (top view)

Bibliography

[1] J. Rabaey, R. Brodersen, A. Stölzle, S. Narayanaswamy, D. Chen, R. Yu, P. Schrupp. H. Murveit, and A. Santos. A Large Vocabulary Real Time Continuous Speech Recognition System. In *VLSI Signal Processing III*, pages 61–74. IEEE Press, 1988.

[2] H. Murveit, J. Mankoski, J. Rabaey, R. Brodersen, A. Stölzle, D. Chen. S. Narayanaswamy, R. Yu, P. Schrupp, R. Schwartz, and A. Santos. A Large-Vocabulary Real-Time Continuous-Speech Recognition System. In *Proc ICASSP 89: 1989 International Conference on Acoustics Speech and Signal Processing*, pages 789–792, May 1989.

[3] A. Stölzle, S. Narayanaswamy, K. Kornegay, J. Rabaey, and R. Brodersen. A VLSI Wordprocessing Subsystem for a Real Time Large Vocabulary Continuous Speech Recognition System. In *Proc CICC 89: 1989 Custom Integrated Circuits Conference*. pages 20.7.1–20.7.5, May 1989.

[4] Electronic Research Laboratory. *LagerIV Distribution 1.0 Silicon Assembly System Manual.* University of California at Berkeley, distribution 1.0 edition, June 1988.

[5] B H Juang. On the Hidden Markov Model and Dynamic Time Warping for Speech Recognition—A Unified View. *AT&T B.L.T.J.*, 63(7):1213–1243, January 1984.

[6] Robert A Kavaler. *The Design and Evaluation of a Speech Recognition System for Engineering Workstations.* PhD thesis, University of California at Berkeley, May 1986.

[7] R. Bisiani, T. Anantharaman, and L. Butcher. Beam: An Accelerator for Speech Recognition. In *Proc ICASSP 89: 1989 International Conference on Acoustics Speech and Signal Processing*, pages 782–784, May 1989.

[8] L R Rabiner and B H Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[9] G D Forney Jr. The Viterbi Algorithm. *Proc. IEEE*, 61:268–278, March 1978.

[10] Walter Scott, Robert Mayo, Gordon Hamachi, and John Ousterhout. *1986 VLSI Tools: Still More Work by the Original Artists.* University of California at Berkeley, December 1985.

[11] VLSI/CAD Group. *THOR.* Stanford University, release 3.2 edition, 1986.

[12] Andrei Vladimirescu and Sally Liu. The Simulation of MOS Integrated Circuits Using SPICE2. Technical Report UCB/ERL M80/7, University of California at Berkeley, February 1980.

[13] Andrew J Burstein. A 9 Bit 10 MHz A/D Macrocell. Master's thesis, University of California at Berkeley, November 1987.