# A Multiprocessor V iterbi Decoder Using Xilinx FPGAs

by

**David Chun-Chin Y eh**

A Thesis submitted in conformity with the requirements
for the degree of Master of Applied Science in the
Department of Electrical Engineering,
University of T oronto

© Copyright by David Chun-Chin Y eh 1995

**A Multiprocessor Viterbi Decoder Using Xilinx FPGAs**

David Chun-Chin Yeh

Master of Applied Science, 1995

Department of Electrical and Computer Engineering
University of Toronto

# Abstract

This thesis presents a general description of the method into the construction of a multi-ring cascaded Viterbi decoder. The general description is further augmented by a discussion of the engineering challenges encountered in the implementation of a Reconfigurable Dual-Ring General Cascade Viterbi Decoder called the RACER that functions as a rate 1/4, constraint length 14 Viterbi decoder. Emphasis is placed on the design and optimization of various portions of the decoder with a view towards implementation in technologies ranging from FPGAs to full custom silicon.

Using 37 FPGAs on a seven processor cards and a custom backplane the RACER can achieve a decoding rate of 41 Kbit/s (49 Kbit/s theoretical) when clocked at 16 MHz.

Finally, we discuss the changes that must be made to achieve a 1Mbit/s FPGA board level system and a 6.47 Mbit/s custom chip version of the RACER.

# Acknowledgments

I would like to thank my supervisor , Professor Paul Chow , as well as Dr . Gennady Feygin for their friendship, help and guidance. I am also grateful for the financial assistance provided by Professor Paul Chow .

I would like to thank my parents for their love, encouragement and support throughout my studies. Moreover , I would like to thank my girlfriend, Lily W ong, for her help, support, and understanding.

I am also grateful for all the graduate students and staff at the University of T oronto who provided the assistance to complete my thesis and most importantly , the friendship they provided. I would especially like to thank Fred Aulich, Paul Chow , Duncan Elliott, Rob Jeschke, David Karchmer , Javad Omidi, Peter Pereira, Ali Sheikholeslami, and Ralph Wittig, for their help.

Gratitude also goes to my friends in helping me proofread my thesis and with special thanks going to Nelson Jean, Janice Liu, and Michael Chang.

I would also like to thank the members of my comittee, Professors G. Gulak, D. Lewis and S. Pasupathy for their helpful comments.

Finally , I am grateful to Xilinx Inc, and Motorola Inc for their generosity in providing me many of the components that I required.

# Contents

**List of Figures** vi

**CHAPTER 1 Introduction** 1

1.1 Motivation . . . . . 1
1.2 Objectives . . . . . 2
1.3 Thesis Overview . . . . . 2

**CHAPTER 2 Background** 4

2.1 Convolutional Coding . . . . . 4
2.2 Viterbi Decoding . . . . . 6
2.2.1 The Branch Metric Generator (BMG) . . . . . 8
2.2.2 The Add-Compare-Select (ACS) Unit . . . . . 9
2.2.3 Survivor Management . . . . . 10
2.3 Viterbi Decoding Architectures . . . . . 11
2.3.1 The Big Viterbi Decoder (BVD) . . . . . 11
2.3.2 DECPeRLe-1 System . . . . . 13
2.3.3 General Cascade Viterbi Decoder . . . . . 13
2.4 Summary . . . . . 17

**CHAPTER 3 The Multi-Ring GCVD** 18

3.1 Introduction . . . . . 18
3.2 Timing for the 2n-1 Ring General Cascade Viterbi Decoder . . . . . 19
3.2.1 Single Ring GCVD . . . . . 19
3.2.2 Multi-Ring GCVD . . . . . 22
3.3 Summary . . . . . 25

**CHAPTER 4 RACER** **26**

4.1 Introduction . . . . . 26
4.2 Goals . . . . . 26
4.3 Hardware Description . . . . . 27
4.4 Processor Implementation . . . . . 33
4.4.1 Floorplanning the Processor FPGA . . . . . 35
4.4.2 Resource Optimization . . . . . 37
4.5 Timing for the Cross-point Switches . . . . . 38
4.6 Survivor Management . . . . . 39
4.6.1 Design Issues . . . . . 40
4.6.2 Traceback for the RACER . . . . . 45
4.7 Performance Data . . . . . 48
4.8 Summary . . . . . 50

**CHAPTER 5 Multi-Ring Viterbi Decoders at 1 Mbit/s and Beyond** **51**

5.1 Introduction . . . . . 51
5.2 The 1 Mbit/s RACER II . . . . . 51
5.2.1 Requirements . . . . . 52
5.2.2 Routing . . . . . 52
5.2.3 Processors . . . . . 54
5.2.4 Cross-Point Switches . . . . . 58
5.2.5 Memory . . . . . 58
5.2.6 Layout . . . . . 59
5.2.7 Estimated Cost . . . . . 63
5.3 The RACER III . . . . . 65
5.3.1 Input/Output Requirements . . . . . 65
5.3.2 Die Size . . . . . 66
5.4 Summary . . . . . 67

**CHAPTER 6 Conclusions and Future Work** **69**

6.1 Summary and Conclusions . . . . . 69
6.2 Future Work . . . . . 69

**References** **71**

# List of Figures


Figure 2-1: A (3,1,2) Convolutional Encoder Example . . . . . 5
Figure 2-2: Sample State Diagram for Convolutional Encoder . . . . . 6
Figure 2-3: Trellis Diagram for a (3, 1, 2) convolutional code . . . . . 7
Figure 2-4: Viterbi Example . . . . . 8
Figure 2-5: Algorithm to find the cheapest flight between cities A and B . . . . . 9
Figure 2-6: Sample Connection of ACS for n = 6 . . . . . 12
Figure 2-7: DECPeRLe-1 . . . . . 14
Figure 2-8: A sample GCVD Architecture for n = 4 . . . . . 15
Figure 2-9: Timing for a GCVD with a constraint length = 4. . . . . 16
Figure 3-1: Butterfly connections . . . . . 19
Figure 3-2: Butterfly Network for Viterbi decoder with constraint length = 4 . . . . . 21
Figure 3-3: Timing for a Dual-Ring GCVD with constraint length = 4 . . . . . 23
Figure 4-1: Block Diagram of the RACER . . . . . 28
Figure 4-2: Block Diagram of Backplane . . . . . 29
Figure 4-3: The RACER Backplane . . . . . 30
Figure 4-4: SUN Workstation and RACER configuration . . . . . 30
Figure 4-5: The RACER Daughter Board . . . . . 31
Figure 4-6: Block Diagram of the Daughter Board . . . . . 32
Figure 4-7: Basic Structure of the Processor Chip . . . . . 33
Figure 4-8: Pipelining of the BMG and ACS portions of the Processor Chip for one PE . . . . . 34
Figure 4-9: Floorplan of the Processor Chip in an XC4010 FPGA . . . . . 36
Figure 4-10: Resource Optimization in a 14-Bit Pipelined Adder . . . . . 37
Figure 4-11: Cross-point Switch . . . . . 38
Figure 4-12: Sample timing for Switches assuming latency for processors is 1 . . . . . 38
Figure 4-13: Pseudo-code for the Two-Pointer Traceback . . . . . 41
Figure 4-14: Two-pointer Traceback Example . . . . . 42
Figure 4-15: Example of Read/Write Pointer Collision . . . . . 43
Figure 4-16: Timing for the RACER Traceback Algorithm . . . . . 47
Figure 4-17: Experimental Setup . . . . . 48

Figure 4-18: Performance Comparison for the RACER and BVD . . . . . 50
Figure 5-1: Wiring for a 23 ring Viterbi decoder . . . . . 53
Figure 5-2: Timing to Multiplex the Loading of the Sample Values . . . . . 55
Figure 5-4: Pipelining for half a PE in the RACER . . . . . 56
Figure 5-3: Possible new pipelining for half a PE in the RACER II . . . . . 57
Figure 5-5: Layout for Daughter Boards 1 and 3 to 7 . . . . . 61
Figure 5-6: Layout for Daughter Board 2 . . . . . 62
Figure 5-7: Alternate Configuration for the RACER . . . . . 63
Figure 5-8: Daughter board for SW-PE-SW-PE Configuration . . . . . 64
Figure 5-9: Proposed RACER III Floorplan . . . . . 67

*Chapter* **1**

# Introduction

## 1.1 Motivation

In this age of digital communication, there is a need to transfer information reliably, particularly in satellite and deep space communication systems, which must deal with low signal to noise (SNR) ratio environments. The Viterbi algorithm [Vitr67][Forn73][Lin83] emerged as a leading contender in these applications. The Viterbi decoder provides a maximum likelihood decoding algorithm for a class of error-correcting codes known as convolutional codes. Many Viterbi decoders have been implemented for satellite and deep space communication. To date, the largest Viterbi decoder is the Big Viterbi Decoder (BVD) [Stat88][Onys91][Coll92] built by the Jet Propulsion Laboratory (JPL) for their Galileo space probe. The BVD is extremely expensive both in cost and development time, motivating us to search for a better alternative. This alternative is the Reconfigurable Cascade Multi-Ring (RACER) Viterbi decoder, which uses a very unique cascade architecture to reduce the complexity and thus cost. The architecture has the advantage of linearly increasing complexity as opposed to a quadratic increase for the BVD design.

Without the luxury of time or enormous budgets, some form of technology must be found to implement the RACER that allows for rapid construction at a reasonable cost. Designing the system using custom VLSI or ASICs would take a great deal of time and there is always a

possibility that multiple iterations of the design would be required. SRAM based FPGAs provide a suitable technological alternative for this project. The main reason for using FPGAs is reprogrammability. In contrast to custom VLSI and ASICs where new chips must be produced for different designs, FPGAs allow many different designs to be implemented with the same hardware. This allows various designs to be implemented to either correct mistakes or upgrade previous versions.

## 1.2 Objectives

The objectives of this work are as follows:

1) To determine the scheduling of the Viterbi decoder primitive operations necessary for the construction of any multi-ring General Cascade Viterbi Decoder (GCVD).
2) To design and implement a Reconfigurable Cascaded Multi-Ring Viterbi Decoder, which we will call the RACER. Since the BVD is the largest Viterbi decoder to date, the RACER will have similar functionality.
3) To create software to communicate between the RACER and a SUN Workstation.
4) To compare the RACER with similar Viterbi decoders.
5) To confirm that the multi-ring GCVD architecture works, and to propose how it can be adapted to construct Viterbi decoders that operate at 1 Mbit/s or higher.

## 1.3 Thesis Overview

This thesis is divided into six chapters. Chapter 2 provides some background information to the Viterbi Algorithm and describes its major components. It then focuses on three different Viterbi Decoding architectures: the BVD, the DECPeRLe-1 and the GCVD.

Chapter 3 starts by introducing the timing equations for the single-ring GCVD and then develops general timing equations for the multi-ring GCVD.

Chapter 4 then provides the details on the actual implementation of the dual-ring GCVD called the RACER. Issues relating to layout of the system, processor floorplanning, timing of the cross-point switch and traceback implementations will also be discussed. Performance and experimental results will be provided and compared with those of the BVD.

Having verified the feasibility of the cascade architecture, Chapter 5 proposes a 1 Mbit/s board level Viterbi decoder using FPGAs and then explores the possibility of implementing a multi-ring Viterbi decoder onto a custom chip. The chapter discusses the technology necessary to achieve these goals and suggests possible configurations.

Finally, Chapter 6 provides conclusions and suggests some future work.

*Chapter* **2**

# Background

The Viterbi algorithm is a maximum likelihood decoding algorithm for a class of error-correcting codes known as convolutional codes. The theory of the algorithm is well documented in the literature [Vitr67][Forn73][Lin83]. Therefore, only brief examples of Viterbi decoding will be provided here. A short introduction of convolutional encoding [Lin83] will be given in Section 2.1, which will then be followed by Viterbi decoding in Section 2.2. Section 2.3 reviews some different architectures of Viterbi decoders, which include the Big Viterbi Decoder [Onys91], the DECPeRLe-1 [Kean94], and the General Cascade Viterbi Decoder [Feyg90] [FeygS93].

## 2.1 Convolutional Coding

Convolutional encoders are described by the number of input streams (k), the number of output streams (n), and the number of previous inputs retained in shift registers (known as the constraint length, $\nu$). Figure 2-1 shows a convolutional encoder for an (n, k, $\nu$) = (3, 1, 2) convolutional code. It accepts a k = 1 input, or a binary input in this case, and provides n = 3 outputs. The three outputs are dependent on the present input, as well as two ($\nu$ = 2) previous inputs. The rate of the code R, which is the ratio of the number of input bits shifted in and the number of output symbols, is k/n or 1/3. As shown, the encoder is a linear feedforward shift register, and depending on the type of generator polynomials (g) required, all or some of the outputs of the shift registers are connected to the modulo-2 adders.

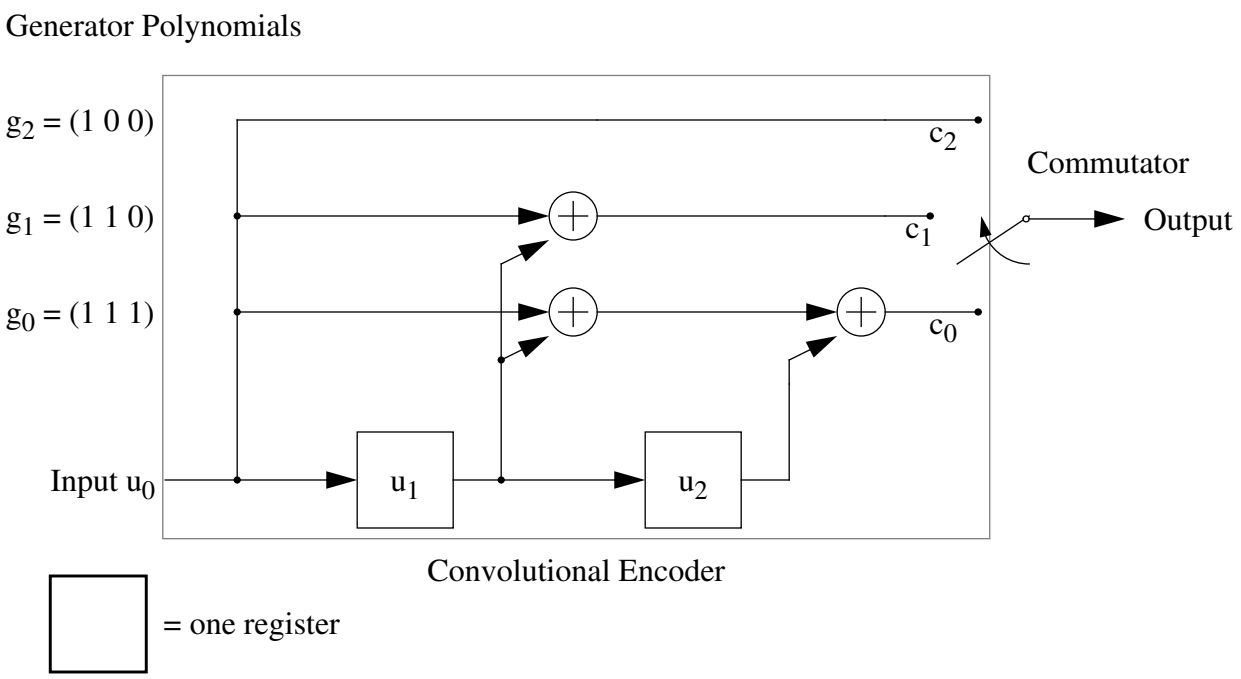



**Figure 2-1:** ***A (3,1,2) Convolutional Encoder Example***

Operation of the encoder begins with the application of an input at $u_0$. This input, along with the previous ν inputs, are then convolved with the generator polynomials $g_0$, $g_1$, and $g_2$. This produces the output codewords $c_0$, $c_1$, and $c_2$, which are sampled by the commutator and sent out in a specific order, such as $c_0c_1c_2$. The registers are then updated and the cycle is repeated with new inputs.

For example, assume that all shift registers are initially zero ($u_1 = u_2 = 0$) and an input of $u_0 = 1$ is applied (Figure 2-1). The values $c_0 = 1$, $c_1 = 1$, and $c_2 = 1$ would be calculated and then sampled by the commutator, providing an output of 111. When the shift registers are updated, the previous contents of $u_2$ are discarded and a new input $u_0 = 1$ is applied. With $u_0u_1u_2$ now changed to 100, the resulting output would be 001. Thus, if the initial conditions of the shift registers are all zero and the pattern $u_0 = (1\ 1\ 1\ 0\ 0)$ is applied at the input, the resulting output, (111 001 101 010 100), would be generated.

The above steps can also be illustrated in a state diagram shown in Figure 2-2, where the nodes represent the present states of the shift registers, and the edges represent transitions to the next state and the corresponding output sequence. In the previous example, the present state would have been 00 and the next state 10 for the initial input $u_0 = 1$. These state transitions can be

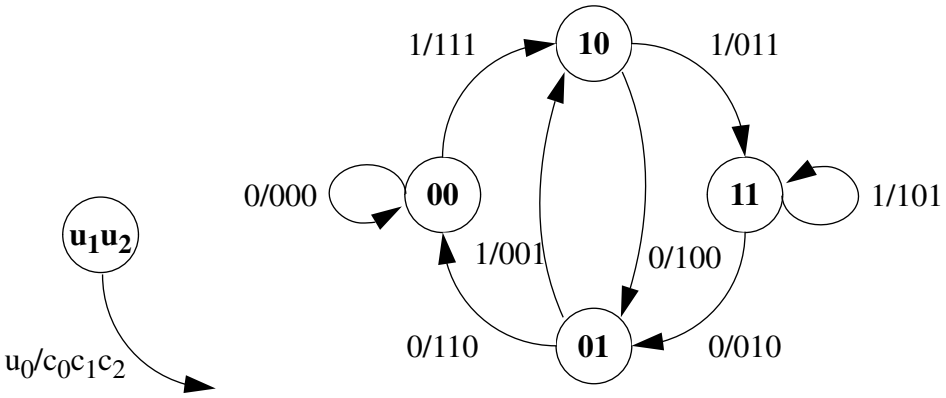
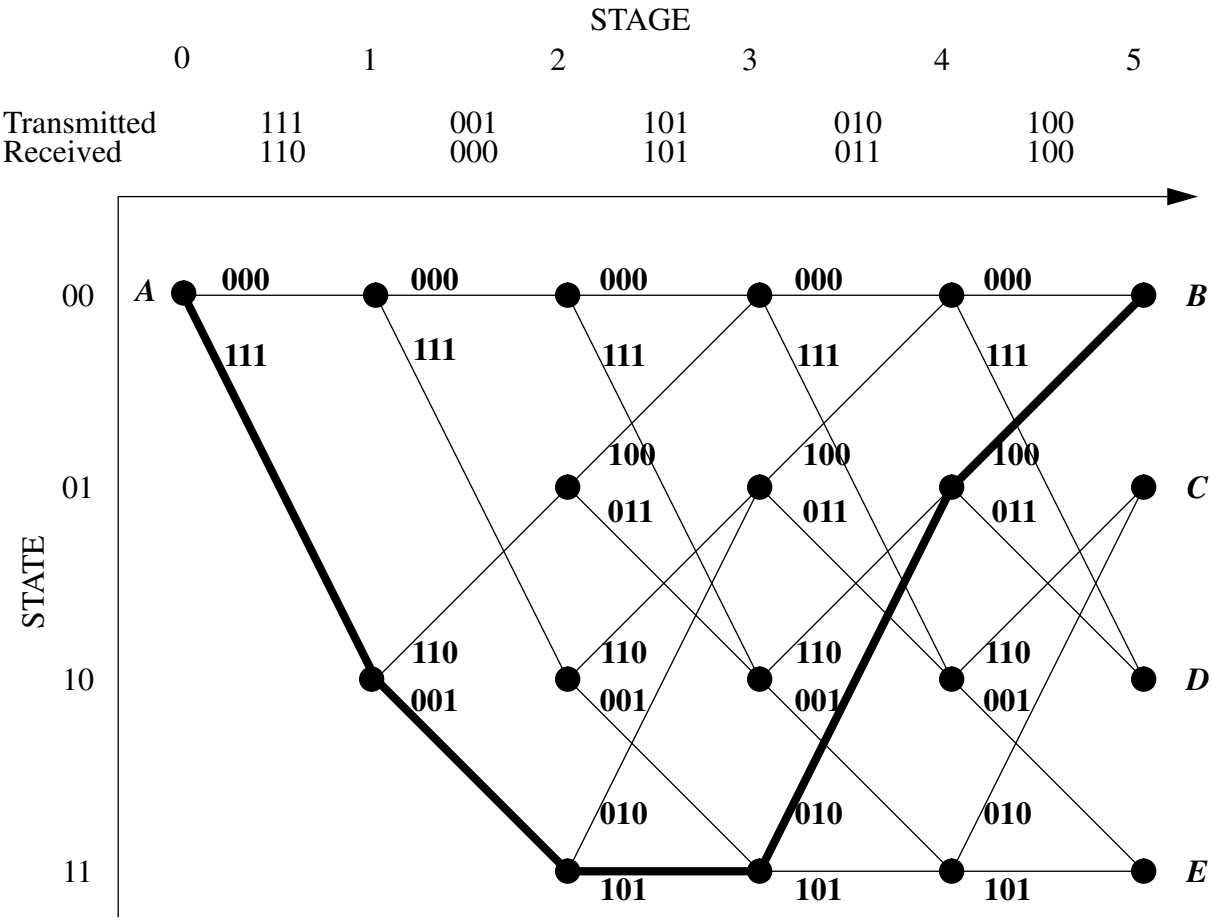



**Figure 2-2:** ***Sample State Diagram for Convolutional Encoder***

expanded in time to what is known as a trellis diagram (Figure 2-3), where the evolution of states in time is shown. The trellis diagram shown assumes that the initial state of the encoder is 00; however, depending on the initial state of the system, the actual starting place may be at any state. The vertical axis shows all the possible states in registers $u_1u_2$ while the horizontal axis represents successive stages of applied inputs to the encoder. Each node represents the present states in registers $u_1u_2$, while the branches indicate a transition to the next state with an applied input $u_0$. An upper branch implies an input of '0' while a lower branch is a '1'. Each path in the trellis diagram represents a unique set of inputs, such as the path highlighted in bold, corresponding to the input sequence $u_0 = (1\ 1\ 1\ 0\ 0)$.

## 2.2 Viterbi Decoding

The process of decoding the information sent by the convolutional encoder is similar to how a travel agent might help a customer find the cheapest flight between cities A and B in the strange world of Binar. This world is unique because it is flat, with all its cities aligned on a grid. To assist in the search for the most economical flight, a map similar to Figure 2-4 is displayed on the agent's terminal. On this map, all arriving and departing flights are shown as lines on the left and right of the cities, respectively. Also, the price of each flight between two cities is shown on the connecting lines between them. With this information, the travel agent proceeds with the algorithm shown in Figure 2-5.




**XXX:** Expected Codewords

**Figure 2-3:** ***Trellis Diagram for a (3, 1, 2) convolutional code***

For example, the minimum total price of the flight to the city located at Stage 4, State 00 would be calculated as follows:

Total price from city at Stage 3, State 00 = price from top flight = 4 + 2 = 6

Total price from city at Stage 3, State 01 = price from bottom flight 5 + 3 = 8

Therefore, the minimum total price is found to be 6 which is then stored at the city located at Stage 4, State 00. An X is marked on the bottom flight indicating that the path should not be taken and an arrow on the top path is marked to point in the correct direction.

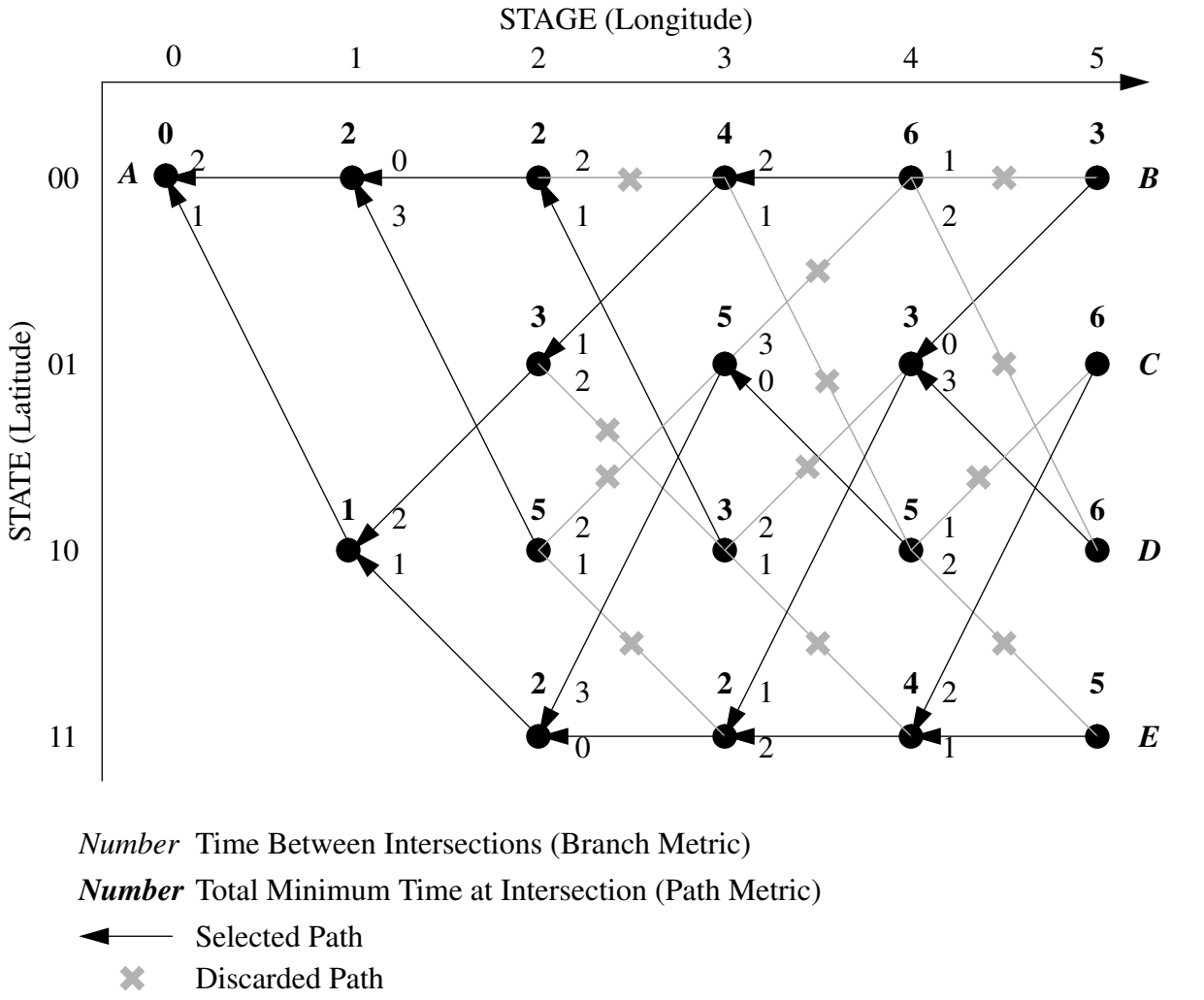



**Figure 2-4:** ***Viterbi Example***

Instead of determining the cost as in the previous example, the main objective for the Viterbi decoder is to determine the path with the least amount of error. To accomplish this goal, the Viterbi decoder has three main components, the Branch Metric Generator (BMG), the Add-Compare-Select Unit (ACS), and the Survivor Management Unit. These components work together for each state to compute the cost of the flight for that stage using the BMG, to calculate the total minimum cost in the ACS, and to store it the best path in the Survivor Management Unit.

### 2.2.1 The Branch Metric Generator (BMG)

As in the case of the travel agent, where they had to know the price between two cities, the BMG calculates all the errors between two stages, which are called branch metrics. To calculate

```
Assume longitude is the horizontal axis and latitude the vertical axis
for longitude = longitude of city A to longitude of city B
    for latitude = 00 to 11
        if city
            if 2 incoming flights
                total price of top flight = total price previous top connecting city + price flying from that city
                total price of bottom flight = total price previous bottom connecting city + price flying from that city
                if total price of top flight = total price of bottom flight
                    total price of flight = total price of bottom flight
                else
                    total price of flight = minimum price (total price of top flight, total price of bottom flight)
                end if
            else
                total price of flight = total price previous connecting city + price flying from that city
            end if
            store this total price of flight at this city
            put an arrow marking the flight path with lowest cost
            put an X marking the flight path with the higher cost
        end if
    end for
end for
following arrows from city B back to city A will give the flight path with the lowest price
```

**Figure 2-5:** ***Algorithm to find the cheapest flight between cities A and B***

this error, the BMG first emulates the convolutional encoder to obtain the expected codewords by convolving the generator polynomials with the present state and the branch to the next state. The error is then calculated by taking the Hamming [Lin83][Feyg90] distance between the expected and received codewords. This Hamming distance is the number of bitwise differences between the two codewords. So for Figure 2-3, the Hamming distance for the upper branch at State 00 Stage 0 is the distance between the expected codeword 000 (in bold) and the received codeword 110 (as indicated above the horizontal axis), which equals two. Figure 2-4 shows the branch metrics for Figure 2-3 assuming the received codewords are as shown in Figure 2-3.

### 2.2.2 The Add-Compare-Select (ACS) Unit

The purpose of the ACS is to find the accumulated minimum error (total price of flight) at a specific state and stage. Information from the BMG (price of flight between intermediate cities)

is sent to the ACS where the accumulated minimum errors from the previous stage, called the path metrics, are added to the branch metrics, their sums compared, and a minimum path metric selected.

Finally, as in the travel agent example, decisions made by the ACS (arrows pointing along the lower cost path) are sent to the Survivor Path Memory Management Unit to be stored and later retrieved to find the survivor path, which is the complete path with the least amount of error. In the example shown above, a traceback method is used to retrieve the decoded values. Other methods, such as the register exchange method, may be used as well. These methods are described in the following sections.

### 2.2.3 Survivor Management

In the travel agent example, there was a fixed starting place and destination; however, what determines the ending of the Viterbi decoder? If there is an infinite stream of inputs arriving in the encoder, the survivor length would become infinite; thus, there must be a point where there is a reasonable assurance that the decoded output is valid. This minimum depth, called the Truncation Depth *T*, is the point where, with a very high probability, all the minimum survivor paths have merged sufficiently to arrive at the same most likely decoded output. Figure 2-4 shows that no matter whether we start from B, C, D, or E, all paths merge at Stage 3, State 11 ($T = 2$). Studies have shown that depending on the type of decoding and the desired Signal-to-Noise Ratio (SNR), truncation depths may vary from as low as 3ν to 10ν or higher if there is a very low SNR [Onys89].

There are generally two methods of storing the survivor sequences: the Register Exchange (RE) method and the Traceback (TB) method. The Register Exchange method is the most common form of storage, while the Traceback method has been used for larger constraint length and low power decoders. Comparisons and detailed descriptions of the Survivor Sequence Memory Management are discussed in the literature [Feyg90][FeygM93]; however, brief descriptions will be presented here.

#### 2.2.3.1 Register Exchange (RE)

Survivor Management by the Register Exchange method is by storage of entire survivor sequences, equivalent to the truncation depth, for each state in the trellis diagram. New values are calculated by the ACS at each state, to be added to the most significant bit (msb) of the survivor sequence and the least significant bit (lsb) is discarded to be used as a possible output of the decoder. The complete survivor sequence is then transferred to the next stage in the trellis diagram and the cycle repeated.

#### 2.2.3.2 Traceback (TB)

The problem with the Register Exchange method is the large bandwidth required to copy the entire survivor sequence between stages. An alternative is to use one bit long pointers indicating an up or down as exemplified in the travel agent example. This eliminates the need to transfer the survivor sequences between stages, reducing the bandwidth required for wiring. When a complete stage is calculated, the Traceback routine will follow the pointers back at least T times after which the data is considered valid decoded data.

## 2.3 Viterbi Decoding Architectures

There have been many different Viterbi architectures, each with their own method of approaching the Viterbi decoding problem. The following sections describe three such architectures for large constraint lengths.

### 2.3.1 The Big Viterbi Decoder (BVD)

The first prototype of Big Viterbi Decoder implemented by Jet Propulsion Laboratory for the Deep Space Network [Stat88] filled two large cabinets and operated at 1 Kbit/s. After verifying the architecture in the first prototype, a second more compact version of the BVD was built. The second BVD [Onys91][Coll92], comprises of 16 identical printed circuit boards with 16 custom VLSI gate-array chips per board, and one board containing the logic for the traceback. All the boards are connected to a 28-layer printed-circuit backplane. It uses 8 Mbit of SRAM for traceback and can attain a data rate of 1 Mbits/s. Connections of the boards and chips are similar to a hierarchical partitioning of a deBruijn graph [Figure 2-6] for $2^{13}$ butterflies, called a crenellated-FFT [Coll88][Coll90]. By using this form of hierarchical partitioning, longer

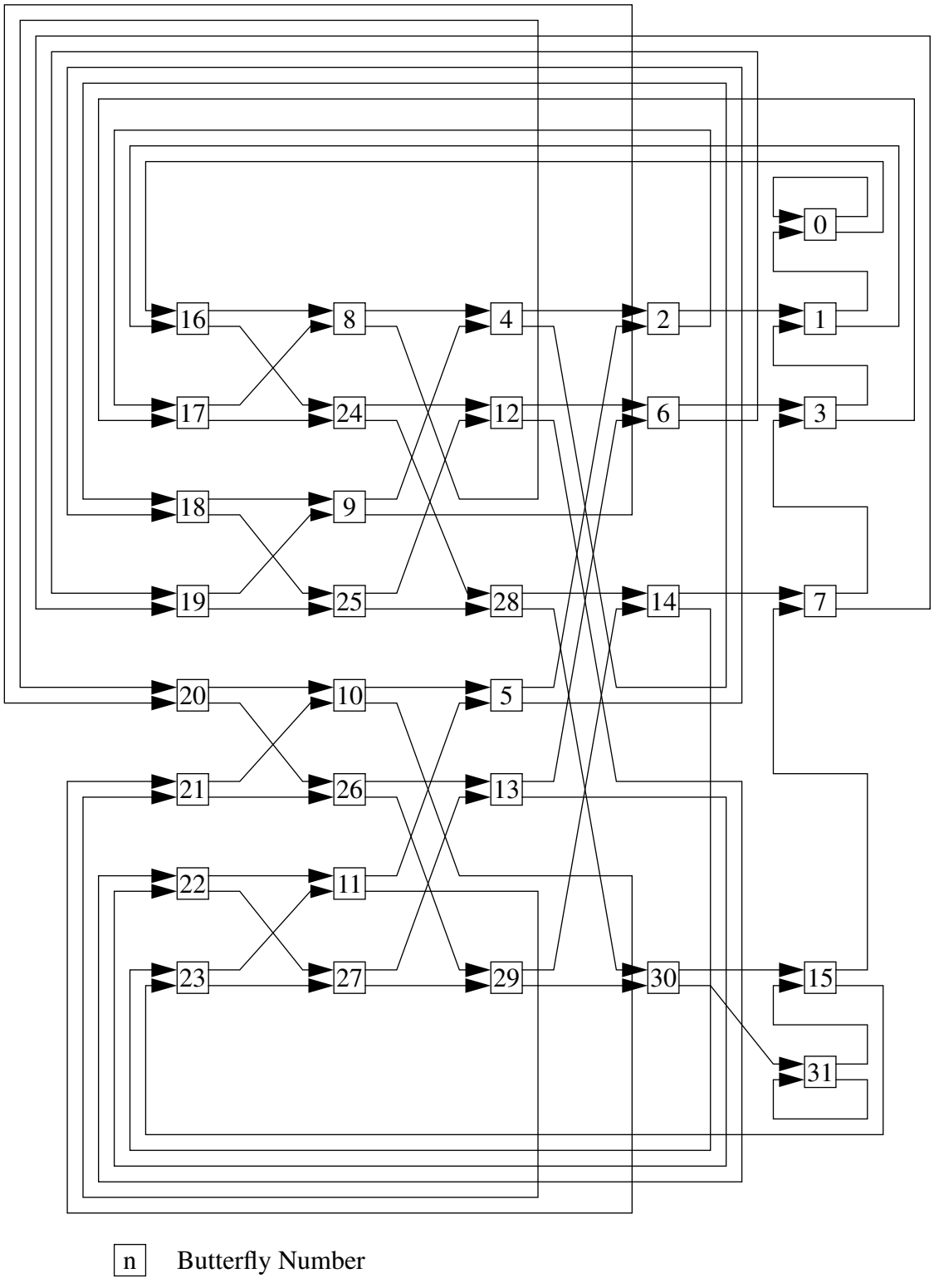



**Figure 2-6:** ***Sample Connection of ACS for*** $\nu = 6$

constraint lengths may be obtained by additional circuit boards. For instance, the present decoder is programmable to process codes with constraint lengths of up to 14 and code rates of 1/2 to 1/6. Increasing the constraint length to 15 would double the number of circuit boards from 16 to 32.

The BVD incorporates a fully parallel approach for decoding information. Due to the massive amount of routing required between processors, all calculations are performed bit-serially.

### 2.3.2 DECPeRLe-1 System

The DECPeRLe-1 [Bert93] is a general-purpose reconfigurable system. It is based on a matrix of 16 Xilinx XC3090 FPGAs connected to four banks of 32-bit by 256K high speed SRAMS (Figure 2-7). Interfacing to the system is accomplished through a DECStation 5000 Ultrix workstation, and programming is done using C++ as a hardware description language. When configured as a code rate 1/4, constraint length 14 Viterbi decoder, the DECPeRLe-1, operating at 25 MHz, can process information at 2 Kbits/s [Kean94].

The system performs the decoding by processing each trellis stage sequentially, calculating four states at a time until all $2^{14}$ states have been determined. Since there are only four ACS processors calculating all $2^{14}$ states, all path metric information must be loaded from RAM and the outputs written back to RAM. A traceback depth of 255 is implemented for the DECPeRLe-1 system.

### 2.3.3 General Cascade Viterbi Decoder

Two problems associated with the above architectures are area and speed. For the BVD, speed was obtained by having one ACS processing element for each of the $2^{13}$ butterflies. The problem with this type of architecture is two-fold. First of all, to route all the path metrics simultaneously to all the processing elements requires an impractical amount of board area; thus, to reduce the amount of routing required, the BVD must process everything bit serially. Executing everything bit serially also prevents the BVD from implementing calculations in a pipelined fashion. The second problem with the BVD is the hardware resources needed for even longer constraint lengths. Increasing the constraint length from 14 to 15 would double the amount of processors needed to decode the data and would quadruple the total area. For the DECPeRLe-1

system, decoding speed is sacrificed for smaller area; since, there are a reduced number of processing elements, more time must be spent calculating the path metric information for all $2^{14}$ states sequentially. Moreover, time must be spent retrieving and updating path metric information.

An alternative to the above architectures is to use a cascade organization [Gulak88]. In this architecture, a ring of processors work in parallel with each processor computing an entire trellis stage. This enables full path metric information to be passed locally from one processor to its neighbouring processor, removing the necessity of bit-serial arithmetic as in the BVD and thus allowing pipelining. Furthermore, unlike the DECPeRLE-1 implementation where an entire trellis stage must be processed before continuing with the next, the cascade architecture permits the computation of multiple stages simultaneously, thereby increasing throughput. In addition, the updating and retrieval of path metric information is no longer required since data is passed locally between stages.

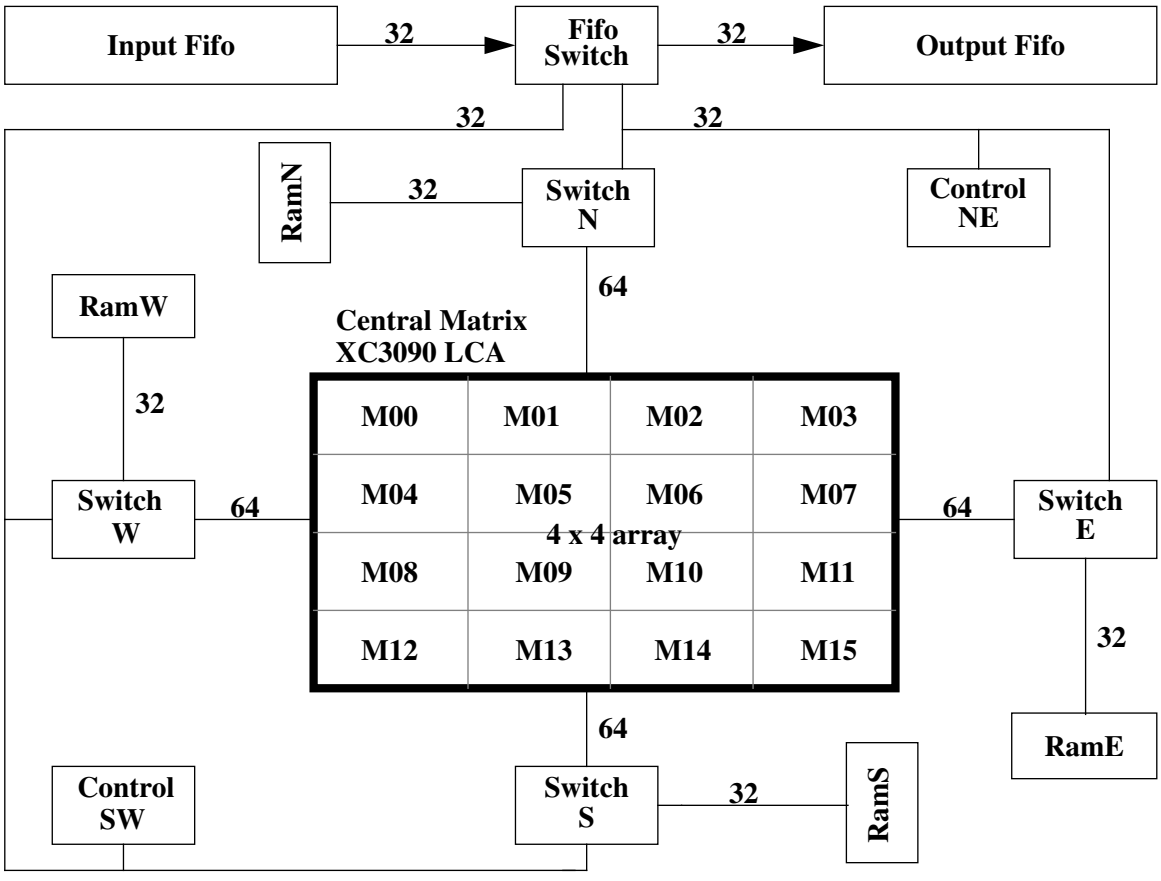



**Figure 2-7:** ***DECPeRLe-1***

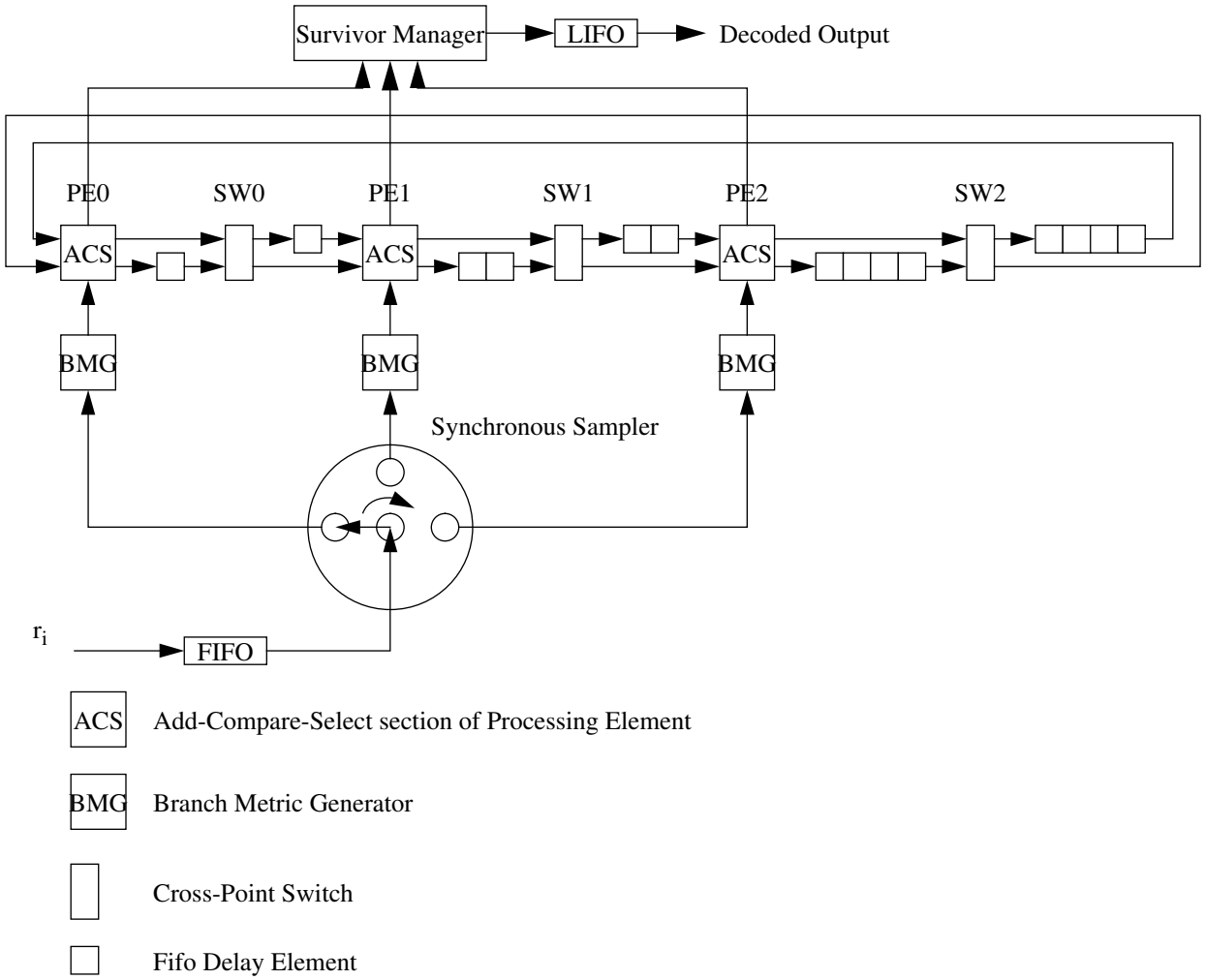



**Figure 2-8:** ***A sample GCVD Architecture for*** $\nu = 4$

A great deal of research has gone into determining the optimum number of processors required to implement the cascade architecture efficiently [Feyg90][FeygS93]. Through this research, it was determined that an arrangement of $\nu$ - 1 processors organized in a ring would provide the best throughput. This architecture is called the General Cascade Viterbi Decoder (GCVD).

The GCVD uses a ring of $\nu-1$ processors to calculate $\nu-1$ stages of ACS computations (Figure 2-8). Path metric information is passed locally from processor k to processor k + 1, with some associated switching. To get a better understanding of the computational flow of such an architecture, Figure 2-9 shows the timing for such a GCVD with a constraint length of four. The numbers shown in the processing elements (PE) and FIFOs represent the states being processed;

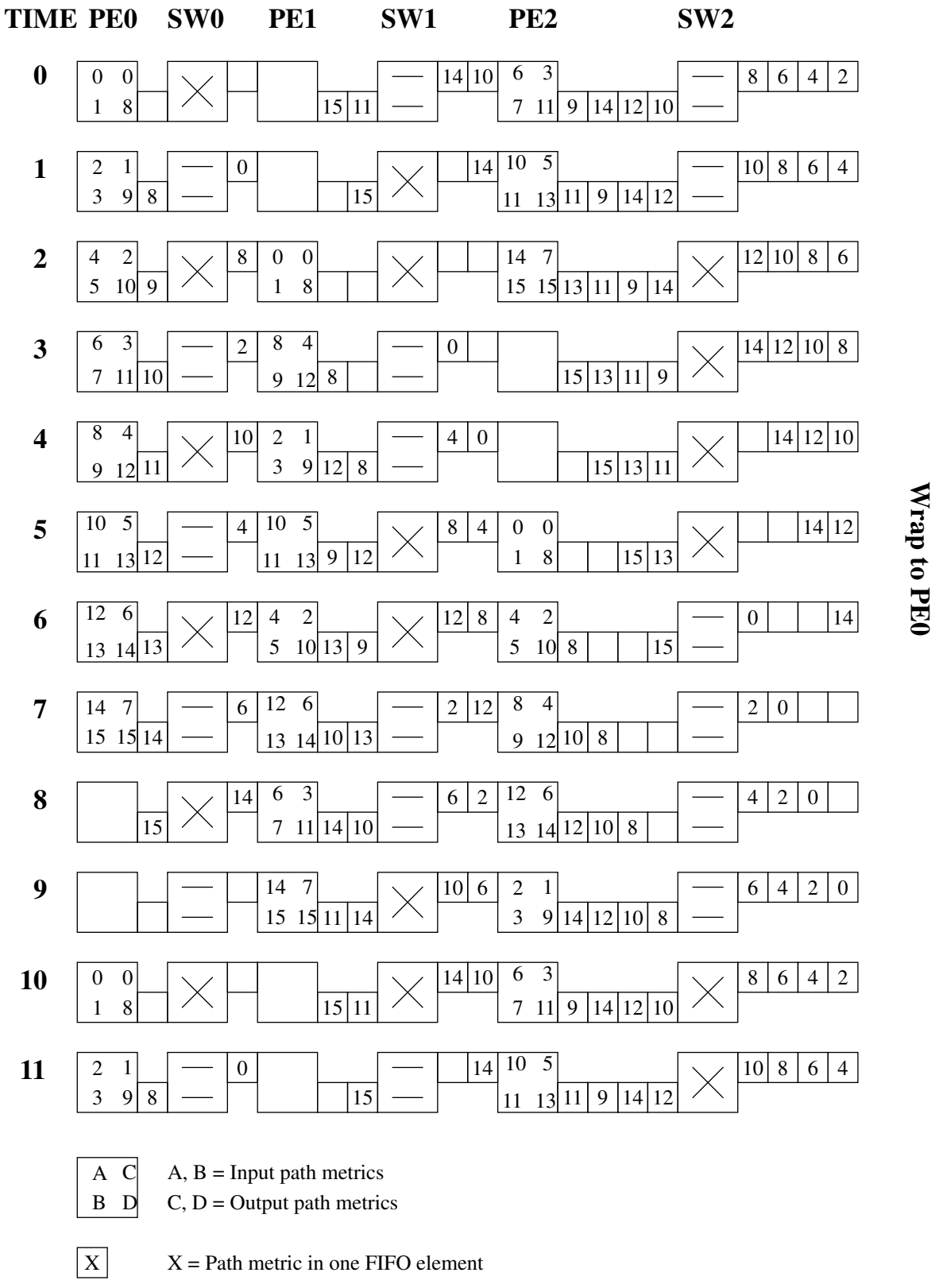



**Figure 2-9:** ***Timing for a GCVD with a constraint length = 4.***

therefore, at time = 0, states 0 and 1 are consumed at the inputs of PE0 and states 0 and 8 of the next stage are determined. The cross shown in the cross-point switch (SW) represents the states being interchanged in the top and bottom, while an equal represents states being passed through. The latency for all processing elements is one. For larger constraint lengths, latency of the PEs may be increased and pipelining employed [Feyg90].

## 2.4 Summary

In this chapter, the concepts of convolutional encoding, Viterbi decoding, and Survivor Management were briefly explained. This was followed by descriptions of several architectures covering different ways of implementing the algorithm. These included the fully parallel and cascade approaches to Viterbi decoding. Finally, we showed that the cascade architecture provided an area-efficient alternative to implementing a long constraint length Viterbi decoder.

*Chapter* **3**

# The Multi-Ring GCVD

## 3.1 Introduction

Field Programmable Gate Arrays (FPGA) play an important role in the prototyping of new designs. Compared to custom ASICs, they allow for quicker implementation of designs at a significantly reduced cost. Moreover, FPGAs are both reprogrammable, which allow designers to quickly correct errors in their prototype, and reusable for other projects. As a result, FPGAs are ideal for this project, due to their cost effectiveness and reusability. However, there are a few drawbacks to using FPGAs. FPGAs currently available are unable to attain the clocking speeds comparable to custom ASICs. This reduced clocking rate lowers the throughput for a comparable design. Consequently, other ways must be found to compensate for this decrease in throughput. In the Viterbi decoder, calculations of the state path metric should be performed as fast as possible. Ways to increase the throughput would include pipelining and/or increasing the number of processors. For the single-ring GCVD explained in the previous chapter, path metrics for two states are calculated simultaneously per processing stage, thus requiring a minimum of $2^{\nu-1}$ clock cycles to complete all $2^{\nu}$ states. However, if a dual-ring GCVD is implemented, four states can be calculated at once, since there will be two processing elements per stage. This reduces the total time to $\frac{2^{\nu-1}}{2} = 2^{\nu-2}$. As more rings are implemented, the completion times for the stages decrease and system clock speeds can be decreased for the same overall throughput. Nevertheless,

as in the fully parallel architecture, the addition of more processor rings increases the need for more wiring resources used for communication between the processors. The remainder of this chapter will describe the schedule required for a $2^{n-1}$ ring processor.

## 3.2 Timing for the $2^{n-1}$ Ring General Cascade Viterbi Decoder

The theory of the single ring General Cascade Viterbi Decoder has been well documented and described in many papers [Feyg90][FeygM93][FeygS93]. Therefore, the definitions and equations derived for the single ring GCVD will be presented but not proven. These equations will be used to derive a new set of equations for the multi-ring GCVD.

### 3.2.1 Single Ring GCVD

#### The Butterfly Representation

Another way of looking at the trellis diagram is through a group of basic elements called butterflies. The one shown in Figure 3-1 represents the add-compare-select computations explained in Section 2.2.2, where a pair of path metric state inputs arrive to derive two output states. For example, the input path metrics for butterfly **100** would be **100**0 and **100**1, and the calculated output path metrics would be for states 0**100** and 1**100**.

#### Local Time and Butterfly Schedule

The number of clock cycles between the start of computations for a particular stage and the time at which a specific butterfly is to be calculated for that stage is called the local time (*lt*).

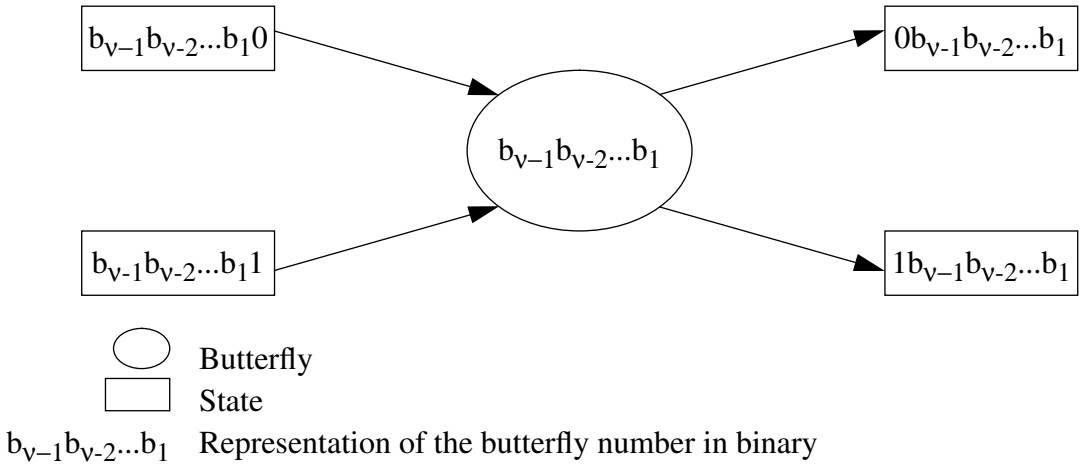



**Figure 3-1:** ***Butterfly connections***

Since all stages in the trellis are uniquely mapped onto the processors numbered from 0 to $j$, where $j = \text{Total \#Stages} \pmod{(\nu - 1)}$, this is also the local time of the processors. Therefore, a schedule for the cascade Viterbi architecture can be defined by the local time at which the inputs of a particular butterfly are required in each processor. The simplest possible schedule for the GCVD is the one where the butterflies in Stage 0 occur in their natural counting order: $0, 1, 2, ..., 2^{\nu-1} - 1$. For this schedule, local time is a circular rotation by $j$ positions of the butterfly number to the left. In particular, in processor $j$, the butterfly $\{b_{\nu-1}b_{\nu-2}...b_1\}$ will occur at:

$$lt = \{b_{\nu-j-1}b_{\nu-j-2}...b_2b_1b_{\nu-1}...b_{\nu-j}\} \quad \text{(EQ. 3-1)}$$

Moreover, for a pipelined processor, *dpe* stages deep, the output of the butterfly will be available at:

$$lt = \{b_{\nu-j-1}b_{\nu-j-2}...b_2b_1b_{\nu-1}...b_{\nu-j}\} + dpe \quad \text{(EQ. 3-2)}$$

The corresponding butterfly network for a constraint length 4 Viterbi decoder is shown in Figure 3-2, with the timing for the single ring GCVD shown in Figure 2-9. For this example the latency of the processors is one while the switches have no delay.

**Interstage Delay**

The interstage delay ($\delta_j$) is the minimum number of clock cycles from the output of the first pair of path metrics in processor stage $j$ to the time at which the first computations are to begin in processor stage $(j+1) \bmod (\nu-1)$ with switches having no delay.

$$\delta_j = 2^j \quad \text{(EQ. 3-3)}$$

With input and output delays in the switches (*dsw*), this changes to:

$$\delta_j = 2^j + dsw \quad \text{(EQ. 3-4)}$$

Referring to the example in Figure 3-3, note that the interstage delay from the $\nu$-2 processor to the 0th processor is the same as the delay from the $\nu$-2 processor to $\nu$-1.

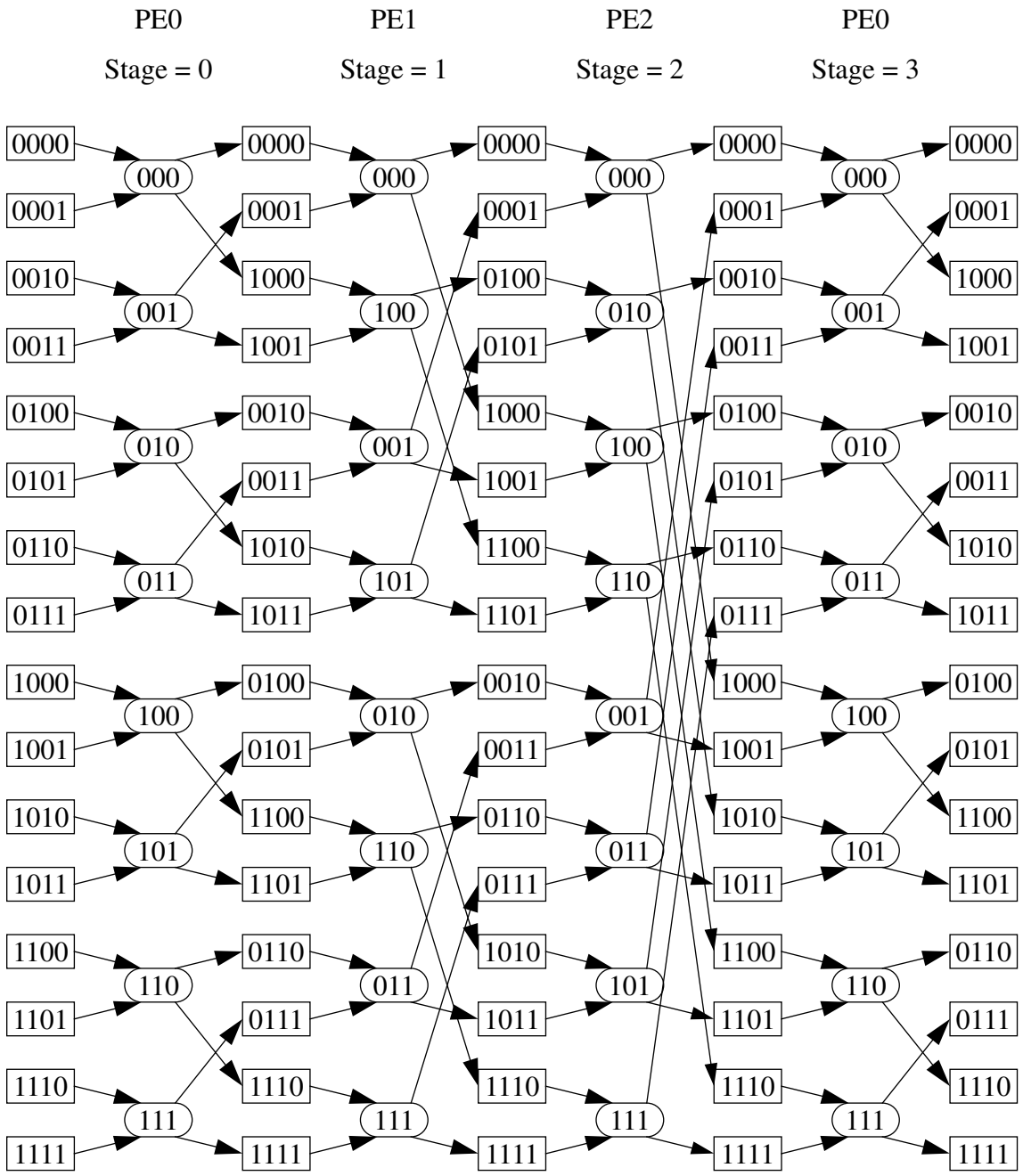



**Figure 3-2:** ***Butterfly Network for Viterbi decoder with constraint length = 4***

**Interprocessor Delay**

The interprocessor delay can be defined as the number of clock cycles between the start of calculations in processors $j$ and $(j+1)\bmod(\nu-1)$. This is simply the interstage delay plus all latency in the processor and associated switches or:

$$\phi_j = \delta_j + dpe = 2^j + dpe + dsw \qquad \text{(EQ. 3-5)}$$

### 3.2.2 Multi-Ring GCVD

The derivation of the multi-ring GCVD is an extension of the single ring GCVD. Instead of one processing element calculating all decisions for a stage, a group of processors, which will now be called a processing stage, will be responsible for a stage. EQ (3-2) showed that the local time for butterfly $\{b_{\nu-1}b_{\nu-2}...b_1\}$ in processor $j$ will occur at $lt = \{b_{\nu-j-1}b_{\nu-j-2}...b_2b_1b_{\nu-1}...b_{\nu-j}\} + dpe$. If another ring of processors is implemented to work in parallel to the first, and if these processors are scheduled properly, the calculation of $2^{\nu-1}$ butterflies would require half the time. If $2^{n-1}$ represents the number of rings present in a Multi-Ring General Cascade Viterbi Decoder then the local time can now be expressed as:

$lt = (\{b_{\nu-j-1}b_{\nu-j-2}...b_2b_1b_{\nu-1}...b_{\nu-j}\} >> (n-1)) + dpe,$ where >> is a bit shift to the right (EQ. 3-6)

Also, $n < \nu$, otherwise the number of processing elements would exceed the number of butterflies for each stage if $n \geq \nu$ and would lead to inefficient use of processor resources.

The multi-ring architecture also influences the interstage and interprocessor delays. In a multi-ring GCVD, more states are calculated simultaneously which allows the processing stages to start earlier than in the single-ring GCVD. Since path metric states are already available in the first few processing stages, they are connected directly. Referring to Figure 3-3, the path metrics between PE0 and PE1 are hardwired directly through SW0 since the states from PE0 are immediately available for PE1 in the next clock cycle. If, for simplicity, the delays for the

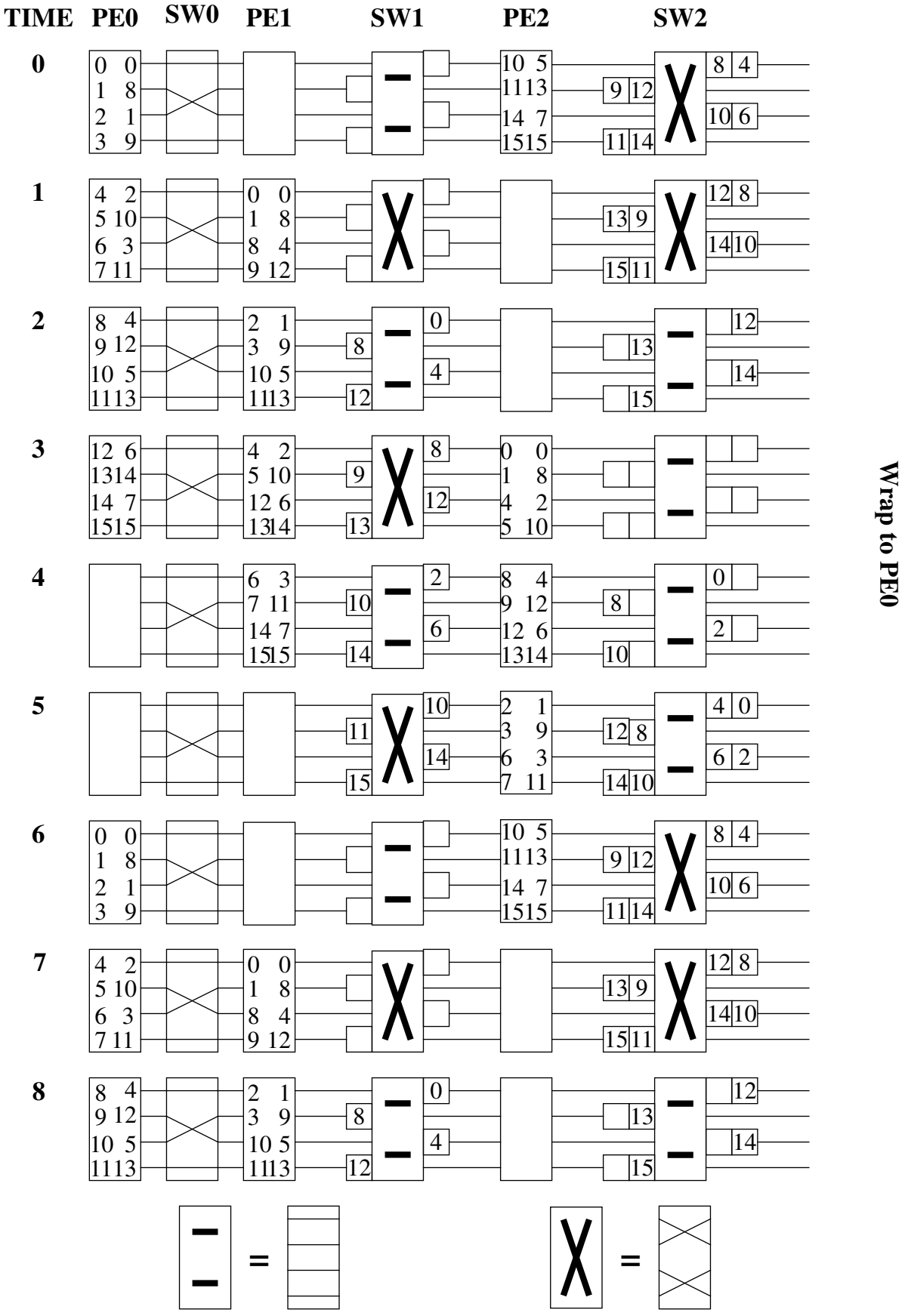



**Figure 3-3:** ***Timing for a Dual-Ring GCVD with constraint length = 4***

hardware connections are identical to the cross-point switches, then the interstage delay and the interprocessor delay for the $2^{n-1}$ ring Viterbi decoder is:

$$\delta_j = \begin{cases} dsw, & j < n-1 \\ \dfrac{2^j}{2^{n-1}} + dsw = 2^{j-n+1} + dsw, & j \geq n-1 \end{cases} \quad \text{(EQ. 3-7)}$$

and

$$\varphi_j = \begin{cases} dpe + dsw, & \mathrm{j} < n-1 \\ \dfrac{2^j}{2^{n-1}} + dpe + dsw = 2^{j-n+1} + dpe + dsw, & \mathrm{j} \geq n-1 \end{cases} \quad \text{(EQ. 3-8)}$$

The total time ($t_j$) at which PEj starts after the initial processing stage PE0 would be the sum of the clock cycles between each successive processing stage, the latency from the processing units, and the latency of the switches. Therefore, using EQ (3-8), the total time is expressed as:

$$t_j = j\,(dpe + dsw), \quad \mathrm{j} < n \quad \text{(EQ. 3-9)}$$

and

$$\begin{aligned} t_j &= (n-1)\,(dpe + dsw) + \left( \sum_{i=n-1}^{j-1} 2^{i-n+1} + dpe + dsw \right) \\ &= (n-1)\,(dpe + dsw) + (j-n+1)\,(dpe + dsw) + \sum_{i=n-1}^{j-1} 2^{i-n+1} \\ &= j\,(dpe + dsw) + 2^{j-n+1} - 1, \quad \mathrm{j} \geq n \end{aligned} \quad \text{(EQ. 3-10)}$$

The number of idle cycles per processing stage ($\Delta$) would be the difference between the total time a processing stage starts at state 0 to where the processing stage must start state 0 again, and the total time required to perform all path metric computations. Using EQ (3-10) for $j = \nu - 1$, the total time to complete one cycle is:

$$t_{\nu-1} = (\nu-1)\,(dpe + dsw) + 2^{\nu-n} - 1 \quad \text{(EQ. 3-11)}$$

The time to calculate $2^\nu$ states is $\dfrac{2^{\nu-1}}{2^{n-1}} = 2^{\nu-n}$. Combining this with EQ (3-11), the number of idle cycles per processing stage is:

$$\Delta = (\nu - 1)(dpe + dsw) + 2^{\nu - n} - 1 - 2^{\nu - n}$$

$$= (\nu - 1)(dpe + dsw) - 1 \quad \text{(EQ. 3-12)}$$

Using EQ (3-12), the percentage of time a processor is doing useful work is:

$$\% \text{ utilization} = \frac{2^{\nu - n}}{2^{\nu - n} + \Delta} \times 100$$

$$= \frac{2^{\nu - n}}{2^{\nu - n} + (\nu - 1)(dpe + dsw) - 1} \times 100 \quad \text{(EQ. 3-13)}$$

The Data Rate for the cascaded multi-ring Viterbi decoder is defined as the number of symbols decoded per second and is given by:

$$\text{Data Rate} = \frac{\text{\# Symbols/Cycle}}{\text{Time to Complete One Cycle}}$$

The number of symbols/cycle is the number of symbols being calculated simultaneously by all processing stages. This is essentially, the number of processing stages in the ring architecture or ν - 1. Therefore, substituting ν - 1 for the #symbols/cycle and EQ (3-11) for the time to complete one cycle, the Data Rate is:

$$\text{Data Rate} = \frac{\nu - 1}{[(\nu - 1)(dpe + dsw) + 2^{\nu - n} - 1] \times T_{clock}} \quad \text{(EQ. 3-14)}$$

where $T_{clock}$ is the system clock period.

## 3.3 Summary

In this chapter, we have derived the timing equations necessary in the development of any multi-ring GCVD. We showed the typical scheduling, the efficiency and the data rates expected for such an architecture. Moreover, with the addition of more rings to the architecture, we demonstrated that the data rate could be increased. The next chapter will describe the development of a dual-ring GCVD implemented as part of this thesis called the Reconfigurable Cascade Viterbi Decoder (RACER).

*Chapter* **4**

# RACER

## 4.1 Introduction

Chapter 3 provided the theoretical details in the implementation of any $2^{n-1}$ ring Viterbi decoder. To demonstrate the viability of the multi-ring GCVD, we have decided to take the next step and migrate the theory into hardware. This chapter will provide details of the implementation of the Reconfigurable Cascade Ring Viterbi Decoder (RACER), which uses a dual-ring GCVD architecture. Its performance will be determined and comparisons will be made against other Viterbi decoders with the same specifications.

## 4.2 Goals

To date, the Big Viterbi Decoder (BVD) built by the Jet Propulsion Laboratory (JPL) for their Galileo space probe is the largest Viterbi decoder available. As a result, its capabilities provide a suitable target for the RACER. The BVD is a rate 1/2 to 1/6, constraint length 14 (16384 states) Viterbi decoder. Not to be outdone, the goals of the project are to build a demonstration Viterbi decoder comparable in functionality to the BVD and to achieve the highest throughput possible with the available technology. Moreover, this Viterbi decoder would be reprogrammable as a simulation engine for any other Viterbi decoder up to and including the size of the BVD, and can be used to implement other algorithms that have a similar ring-like architecture.

Some considerations that are made in the implementation of this project are cost and time. Not only is the simulation, design, and fabrication of many different ASIC chips time consuming, but they are also very expensive. Moreover, first designs are not guaranteed to function correctly; thus, requiring additional time and money to correct any flaws. SRAM based FPGAs resolve some of these difficulties. They are relatively cheap and reprogrammable for the rapid prototyping of designs. Reprogrammability also allows users to correct flaws easily, which avoids the cost of refabrication as in the case of a custom ASIC. However, the shortcomings of FPGAs are slower speeds and lower densities compared to ASICs. Nevertheless, once designs have been verified in FPGAs, pin-compatible custom ASIC chips can be produced. With these advantages, SRAM based FPGAs are used to aid in the prototyping of the RACER. Many types of FPGAs are available; however, the RACER uses Xilinx XC4010-5PQ208 FPGAs [Xlxc93] generously donated by Xilinx Inc.

## 4.3 Hardware Description

From a block diagram of the "Flattened" version of the RACER shown in Figure 4-1, it can be seen that the RACER has a very regular structure. This regularity allows the RACER to be easily partitioned into many smaller sections. Specifically, the RACER is a full custom multiprocessor system broken up into seven daughter boards containing all the processors, switches and RAM controllers, with a backplane to provide communication between the daughter boards. The backplane also provides synchronization and interfacing to the SUN workstation.

Figure 4-2 shows a block diagram of the backplane. It consists of a Xilinx XC4010 FPGA used as a controller for the whole system, two 2Kx18 FIFOs for input buffering, one 1Kx9 FIFO for output buffering, fourteen 144-pin connectors for the daughter boards, one chip for distribution of the clock to all the daughter boards, one 10-pin connector for the programming of the FPGAs, and two 40-pin connectors acting as I/O between the RACER and the interface board. Note that the slots for the daughter boards are interleaved. This is to allow the shortest average distance between two consecutive daughter boards when they are connected in a ring. The physical layout is shown in Figure 4-3. The backplane is made up of six layers, four for signals, one for power and one for ground.

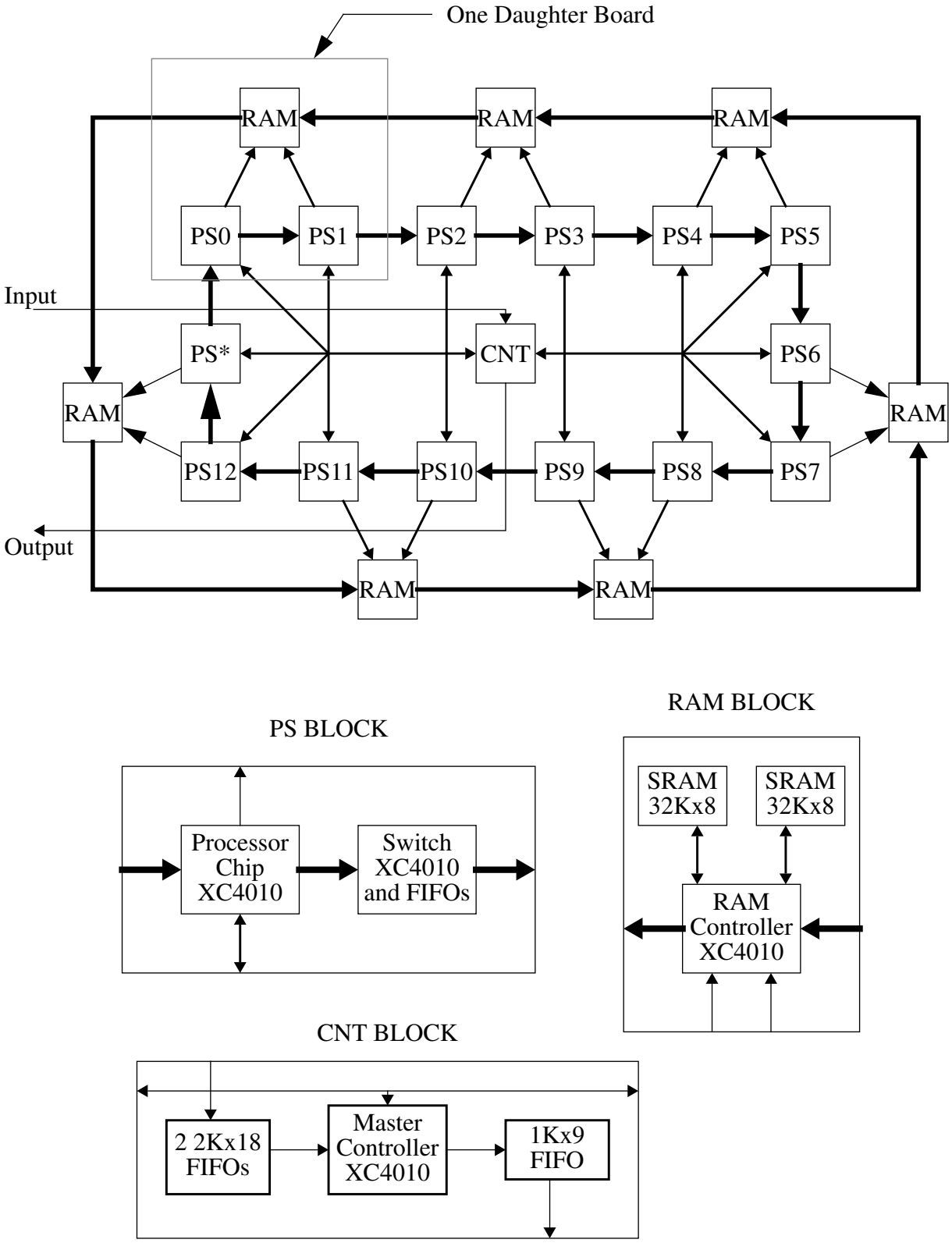



**Figure 4-1:** ***Block Diagram of the RACER***

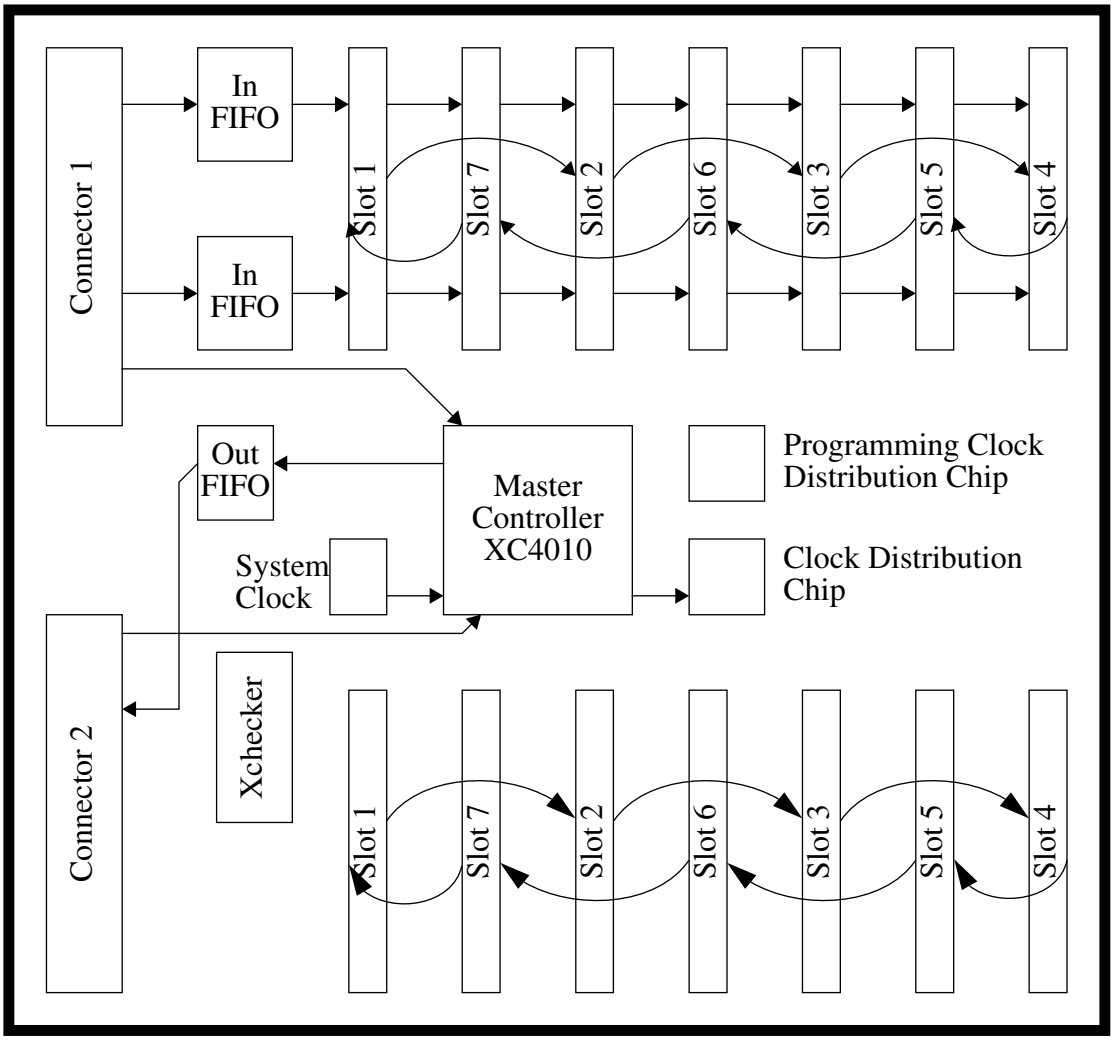



**Figure 4-2:** ***Block Diagram of Backplane***

Communication between the SUN workstation and the RACER is through an SBUS interface board built for the Transmogrifier-1 (TM-1) [Gall94]. Connector 1 is used to send the input samples and the external clock from the SUN workstation to the RACER while all the control signals are sent through Connector 2. The configuration is shown in Figure 4-4.

The physical layout of all the daughter boards (Figure 4-5) are identical except for the number of components mounted. Again, six layers are used to implement the board: four for signal, one for power and one for ground. As shown in the block diagram (Figure 4-6), each daughter board contains five XC4010 FPGAs, one clock distribution chip for fan-out to all components on the daughter board, two 32Kx8 SRAMs, space for eight 2Kx18 FIFOs and two 144-pin connectors. Recall that as the constraint length grows, the size of the FIFOs required for

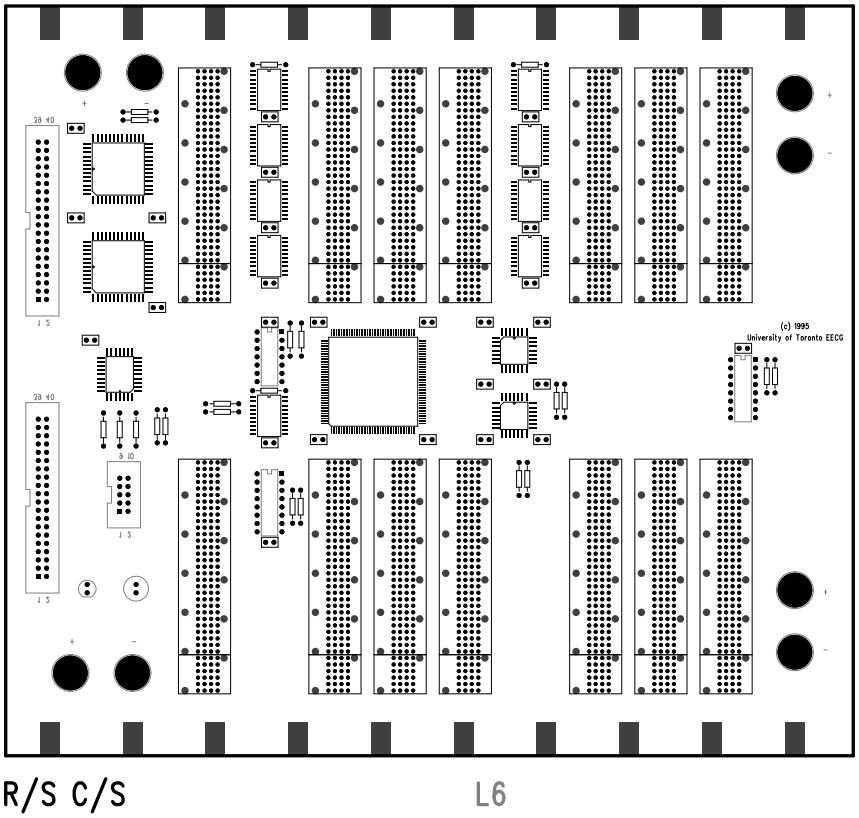



**Figure 4-3:** ***The RACER Backplane***

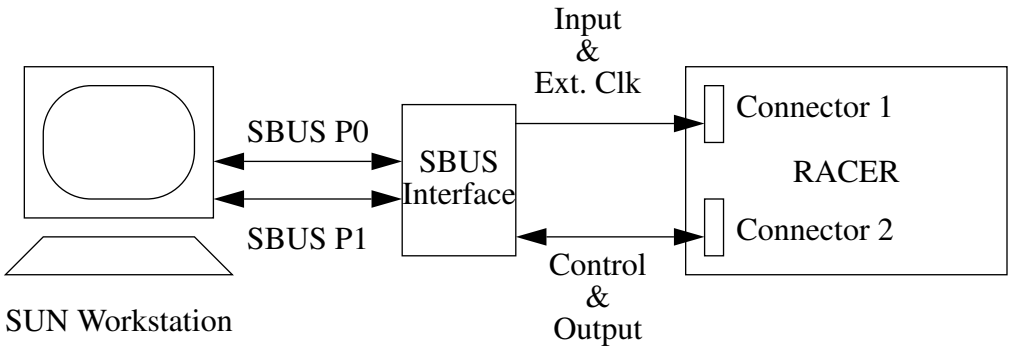



**Figure 4-4:** ***SUN Workstation and RACER configuration***

reordering of the path metric grows as well. With most of the resources available in the FPGAs implementing the cross-point switches, the small FIFOs can be included into the FPGAs. However, if the FIFO size exceeds four 128x14 FIFOs, external components must be used.

To keep the system cost low, the layout of all daughter boards should ideally be identical; nevertheless, with different FIFO requirements and the fact that some FIFOs could be efficiently implemented in the FPGAs, a design problem arises. One solution would be to have eight external 2Kx18 FIFOS on every board and forgo the chance to implement some FIFOs in the FPGAs. This solution does solve the layout problem but incurs a cost of 40 additional FIFOs that could have been included in the FPGAs. A better alternative, which was chosen for the project, is to create the component footprints of the FIFOs with their inputs and corresponding outputs shorted together. Without any modifications to the board, path metric information would pass freely between FPGAs; however, in those instances where external FIFOs are required, the shorts on the footprints would be manually removed and FIFOs placed in those positions.

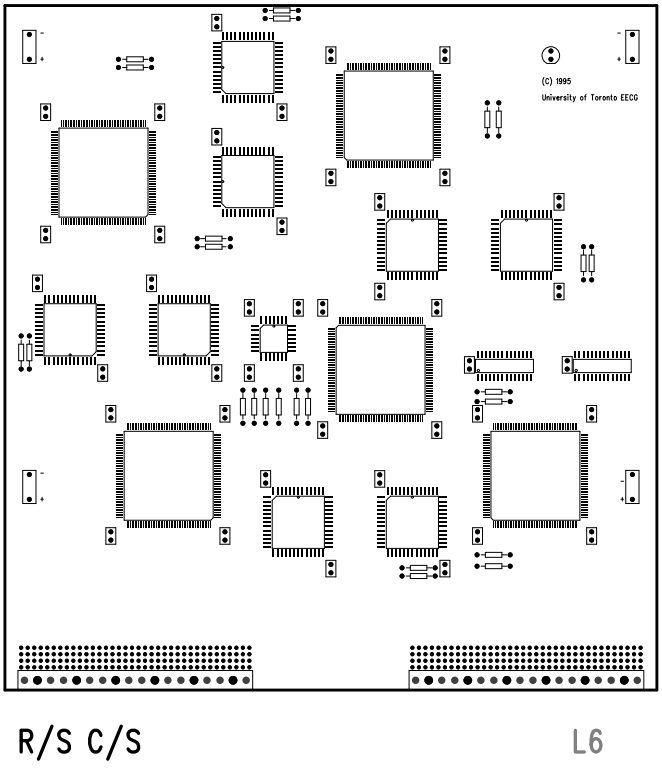



**Figure 4-5:** ***The RACER Daughter Board***

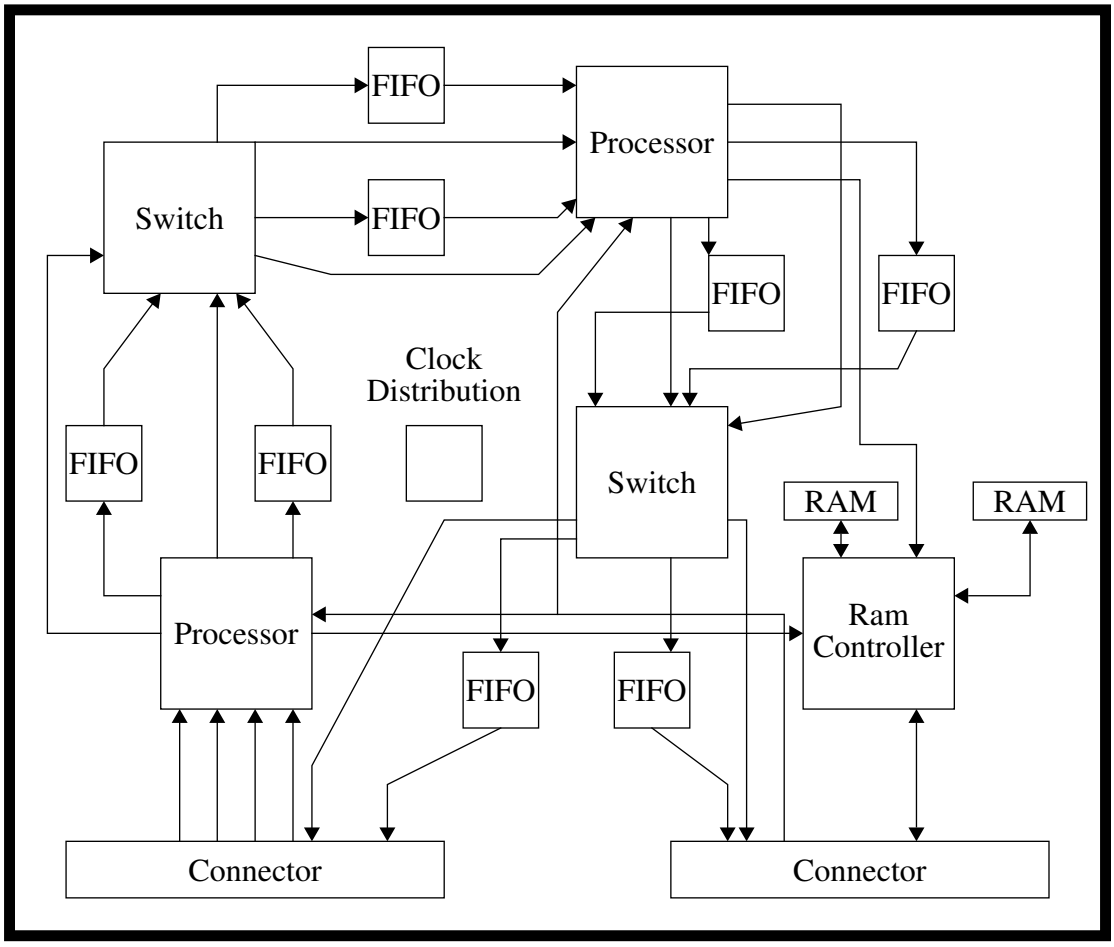



**Figure 4-6:** ***Block Diagram of the Daughter Board***

Ideally, the final version of the RACER should be implemented as a single board design since it would provide a substantial cost savings. The savings would mostly be in not requiring the high pin count connectors and a more efficient board layout. Moreover, the board could be made faster and more reliable, since connectors would no longer be required to connect the long traces between daughter boards. However, testability is a major issue for any new design. For a prototype system of this magnitude, debugging could be made simpler by taking a divide and conquer approach. If a large board could be partitioned efficiently into many smaller sub-boards, then each sub-board could be tested individually. Therefore, we have decided to implement the RACER as a multiple board architecture.

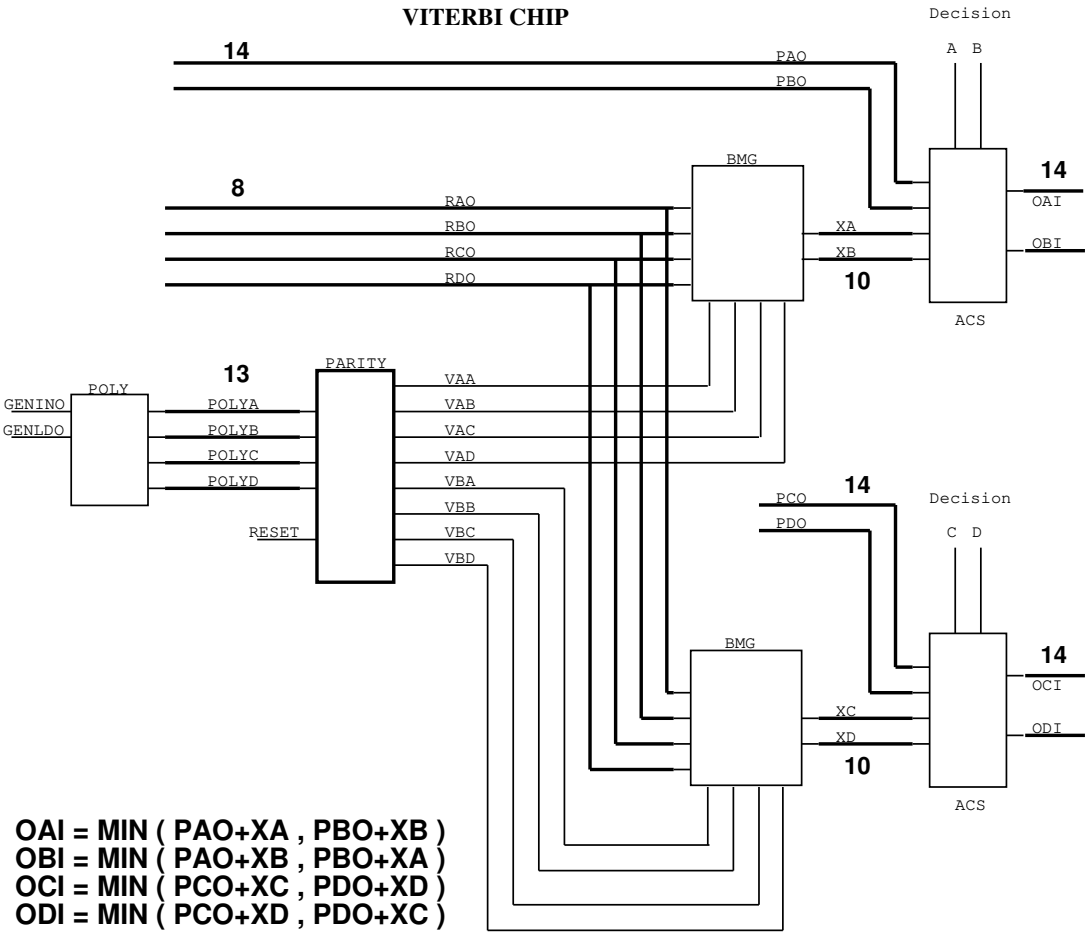



**Figure 4-7:** ***Basic Structure of the Processor Chip***

## 4.4 Processor Implementation

The critical path of the RACER is in the path metric update section of the processor; therefore, its implementation in the Xilinx XC4010 FPGA is optimized to operate as fast as possible. As stated before, a dual-ring architecture is implemented to improve throughput, meaning each processor chip contains two processing elements (PE). As specified in Gennady Feygin's thesis [Feyg90], one half of each PE performs one ACS calculation by taking a pair of 14-bit path metric values, adding them to a pair of 10-bit branch metric values generated by the BMG, comparing the two results and selecting the smaller value. The basic structure of the processor chip is shown in Figure 4-7, and the pipelining of the BMG and ACS portion of the chip is shown in Figure 4-8. As shown in Table 4-1, over 83% of the XC4010 FPGA resources are used. This high utilization of FPGA resources poses a few problems. The primary problem is the ability of the Xilinx *ppr* [Xlxa91] (place and route) program to place and route the entire chip. As

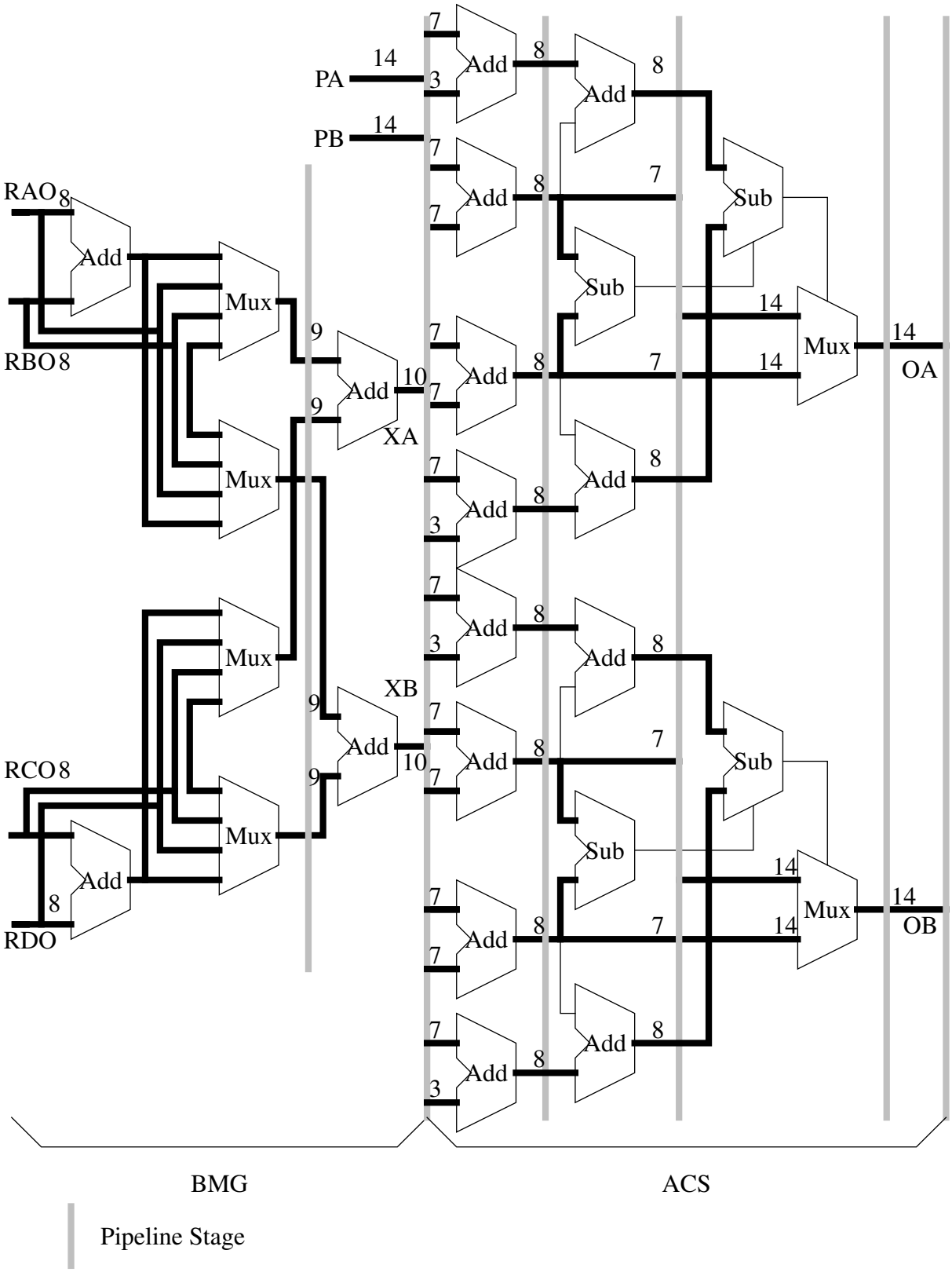



**Figure 4-8:** ***Pipelining of the BMG and ACS portions of the Processor Chip for one PE***

shown, the processor chip is mostly datapath in nature, and it would be best if the logic in the datapath were placed in a regular manner similar to a custom VLSI implementation. If the datapath is specified as a netlist derived from a schematic definition, the netlist can be given to the place-and-route software to try and find a suitable logic mapping and placement for each CLB. However, due to the nature of place and route algorithms, the result will be very random in nature, losing most of the regularity of the structure that is available because the function is actually a datapath. This significantly decreases the number of logic functions that can be put inside the FPGA. The next section discusses the technique that was used to fit the circuit within a single XC4010.

| | |
|---|---|
| 98% Utilization of I/O Pins | 156 of 160 |
| 83% Utilization of function Generators | 665 of 800 |
| 85% Utilization of Clb Flip-Flops | 682 of 800 |

**Table 4-1:** ***Utilization of FPGA before routing***

### 4.4.1 Floorplanning the Processor FPGA

To try and preserve the regularity of the datapath, a different strategy is used. Instead of allowing the circuit to be decomposed into functions that are mapped at the CLB level, we define macros that consist of functional slices of the datapath. Hard-macros [Xlxb91] are good in this case since they contain complete functions that have a predetermined mapping to the CLBs. The use of macros parallels how a custom VLSI datapath might be built. For example, one slice might be a complete nine-bit adder, and another might be an eight-bit wide set of 2:1 multiplexers with registered outputs. Once the hard-macros are defined, they are manually placed exactly in the order they are to appear in the datapath. This leaves the router little choice in that the only routing necessary is to go from one hard-macro to the immediately neighbouring hard-macro, maximizing the usage of the available resources. Random control logic that cannot be made into hard-macros is allowed to be placed and routed automatically.

Two ways of manually placing macros are either by schematic entry, where the placement is defined on a drawing, or by an external file that is read by the place and route program. An external file is preferred since modifications to the placement could be performed by using a text

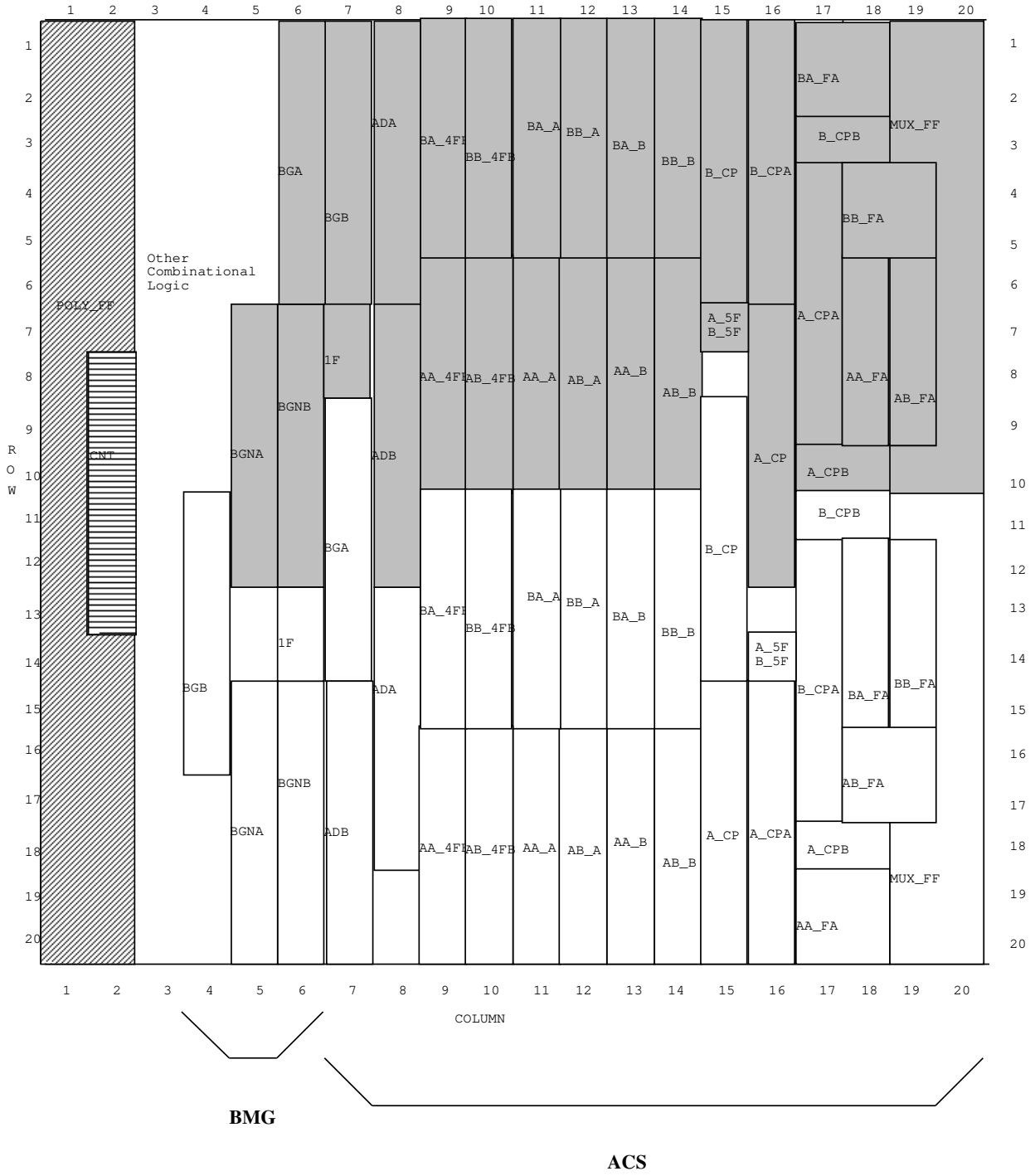



**Figure 4-9:** ***Floorplan of the Processor Chip in an XC4010 FPGA***

editor rather than by direct editing of the schematic. For this particular design, a file with the extension of *.cst* was made to be read by the *ppr* program. Figure 4-9 shows the placement of the macros with the solid grey and white regions representing the two processing elements.

### 4.4.2 Resource Optimization

An important part in implementing a design in a specific technology is to understand the technology. FPGAs are no exception. An example is pipelining a 14-bit adder into two 7-bit adders. From Figure 4-10, it may seem that design (a) would require fewer CLBs than design (b) since it does not require the extra adder; however, upon closer inspection, design (b) actually utilizes fewer CLBs. This is due to the way Xilinx FPGAs are constructed. In Xilinx FPGAs, each CLB contains two 4-input look-up tables and two flip-flops. If there is no combinational logic preceding the flip-flops, then the look-up tables cannot be used for any other purpose. Therefore, design (b) provides a lower CLB count since by using the previous unusable look-up tables to implement a 7-bit adder, design (b) is able to merge the two 7-bit flip-flops into one 8-bit flip-flop reducing the CLB count by two. This technique was also used to reduce the CLB count in the RACER processor design.

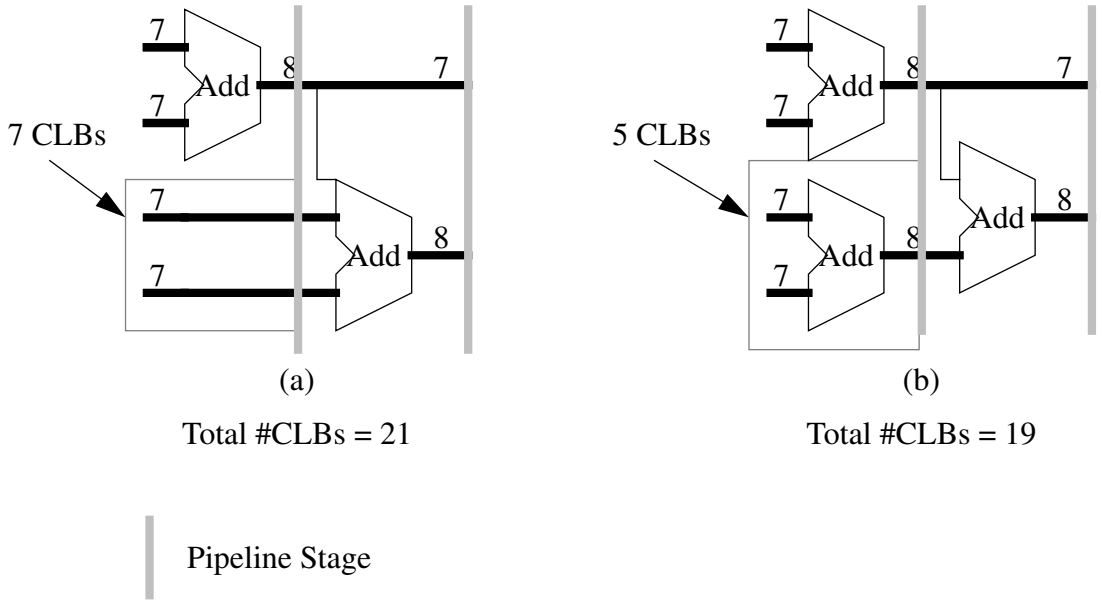



**Figure 4-10:** ***Resource Optimization in a 14-Bit Pipelined Adder***

## 4.5 Timing for the Cross-point Switches

The purpose of the cross-point switch [Feyg90] is to provide reordering of the path metric states such that they arrive at the input of the processing element at the appropriate time and at the correct (upper or lower) input of the following stage. Referring to Figure 4-11, the cross-point switch is programmable to provide a delay of 0 or $2^i$ to the upper input and $2^i$ or $2^{i+1}$ to the lower input. As an example, referring to Figure 4-12, which is a portion of Figure 2-9, if PE1 is to start in the correct order, the top output states of PE0 at Times 0 and 1 must be delayed by 1 and 0

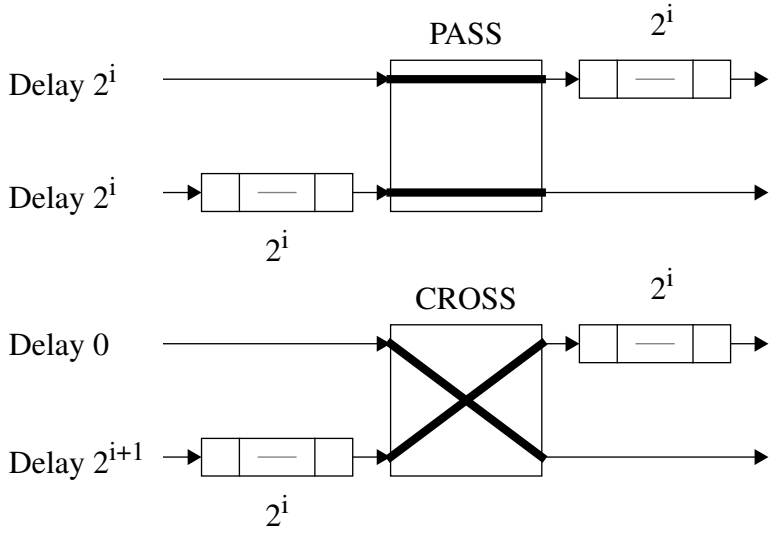



**Figure 4-11:** ***Cross-point Switch***

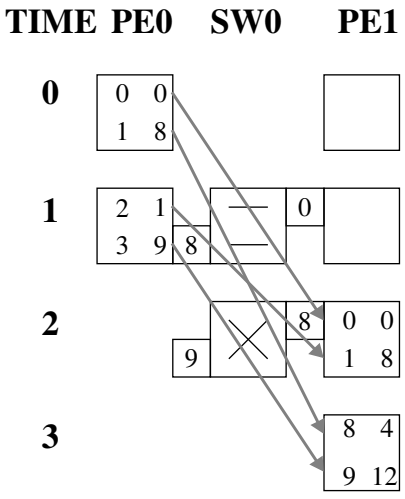



**Figure 4-12:** ***Sample timing for Switches assuming latency for processors is 1***

respectively. Moreover, the bottom outputs states must be delayed by 2 and 1. The delays for each processing stage for a single-ring GCVD is $2^j$. As more rings are used, the processing stages start earlier and as a result, the depth of the delays at each processing reduces to:

$$\frac{2^j}{2^{n-1}} = 2^{j-n+1}, \quad j \geq n-1 \qquad \text{(EQ. 4-1)}$$

The switch starts at pass and alternates to cross and pass, *dpe* + *input delay of switch* after the start of the processing stage. The number of cycles the switch stays at pass or cross is expressed the same as EQ (4-1).

## 4.6 Survivor Management

Efficient storage and recovery of the decoded information is also very important in Viterbi decoding. As discussed in Section 2.2.3, there are generally two methods of survival management with preference going to the Traceback (TB) method for long constraint lengths [Feyg90][FeygM93]. The following will be a discussion on the implementation of the Traceback method for the RACER with an emphasis on the Two-pointer Traceback Algorithm [Feyg90][FeygM93].

Recall that an ideal survivor management algorithm would have an infinite amount of memory to store all the survivor sequences. Since infinite memory is impractical, the alternative is to incorporate enough memory to ensure data integrity. This amount of memory or the depth at which the survivor path must be kept is called the Truncation Depth (T).

As the name implies, the Two-pointer Traceback Algorithm uses two pointers, a read pointer and a write pointer. Recall that to arrive at the correct decoded information, T survivor stages must be traced through; therefore, the read pointer performs two functions: a Traceback Read and a Decode Read. In the Traceback Read, the read pointer first reads the data from the state it is pointing to and then proceeds to determine the new state of the next stage. This new state is determined in the following manner. Assume that the bit read from the present state $\{S_3S_2S_1S_0\}$ is X, then the next state to be read is $\{S_2S_1S_0X\}$. The procedure is repeated until the read pointer moves T stages back. After executing the Traceback Read, we arrive at the point where all decisions read after this point are the decoded information bits; thus, the Decode Read

may proceed. The Decode Read functions in the same manner as the Traceback Read except that all decisions are now regarded as decoded output. Moreover, since the Decode Read proceeds in the opposite direction to the data transmission, the decision bits are in reverse order. To reorder the decision bits, a last-in-first-out (LIFO) buffer is required. In parallel to the read pointer, the Write pointer moves in the opposite direction, keeping track of the stage on the trellis that is currently being written. A pseudo-code of the algorithm is shown in Figure 4-13.

As shown in Figure 4-14, a good analogy for the operation of the pointers are two wavefronts converging in opposite directions. In this example, both pointers proceed simultaneously. If all RAM banks are the same size, as shown in the diagram, the Read Pointers would ideally be operating at twice the speed of the Write Pointers such that both pointers may proceed without stalling. This is easily accomplished since many states must be written per stage during the time that one bit needs to be read.

### 4.6.1 Design Issues

The previous Traceback example works very well for the fully parallel approach to Viterbi decoding; however, with all the processing stages starting at different times in the multi-ring GCVD, overlap of the Decode Read and Write pointers could occur as shown in Figure 4-15. One solution to the problem would be to add buffers to the write decisions such that all stages complete at the same time and to allow time for the decode pointer to complete. Since PEν-2 is the last processing stage to start, the largest buffer would be required at PE0 since it is the first processing stage. Using EQ (3-10), the number of cycles between PE0 and PEν-2 is $t_{\nu-2}$. Therefore, for the RACER, with $dpe + dsw = 7$, n = 2 and ν = 14, the number of cycles between the two processing stages is:

$$t_{12} = 12 \times 7 + 2^{12-2+1} - 1 = 2131 \text{ cycles}$$

Since there are four decision bits per cycle, the largest buffer size is $2131 \times 4 = 8524$ bits. This is approximately 2/3 of the total memory in the XC4010. However, with control logic and routing, the buffer is too large resulting in the need for external FIFOs.

```
T = 10ν
read_state = 0
write_pointer = 0
read_pointer = 3T
read_type = traceback
stage_count = 0


Main Procedure
        Simultaneously do the Write Procedure and Read Procedure
End Procedure


Write Procedure
        for state = 0 to 2^ν-1
                write to RAM[write_pointer][state]
        end for
        write_pointer = (write_pointer+1) mod 3T
End Procedure


Read Procedure
        For i = 0 to 1
                decision_bit = read from RAM[read_pointer][read_state]
                if read_type = traceback
                        send decision_bit to LIFO
                end if
                read_state = left_shift(read_state) + decision_bit
                read_pointer = (read_pointer - 1) mod 3T
        end for
        stage_count = (stage_count + 2) mod T
        if stage_count = 0
                if read_type = traceback then
                        read_type = decode
                else
                        read_type = traceback
                end if
        end if
End Procedure
```

**Figure 4-13:** ***Pseudo-code for the Two-Pointer Traceback***

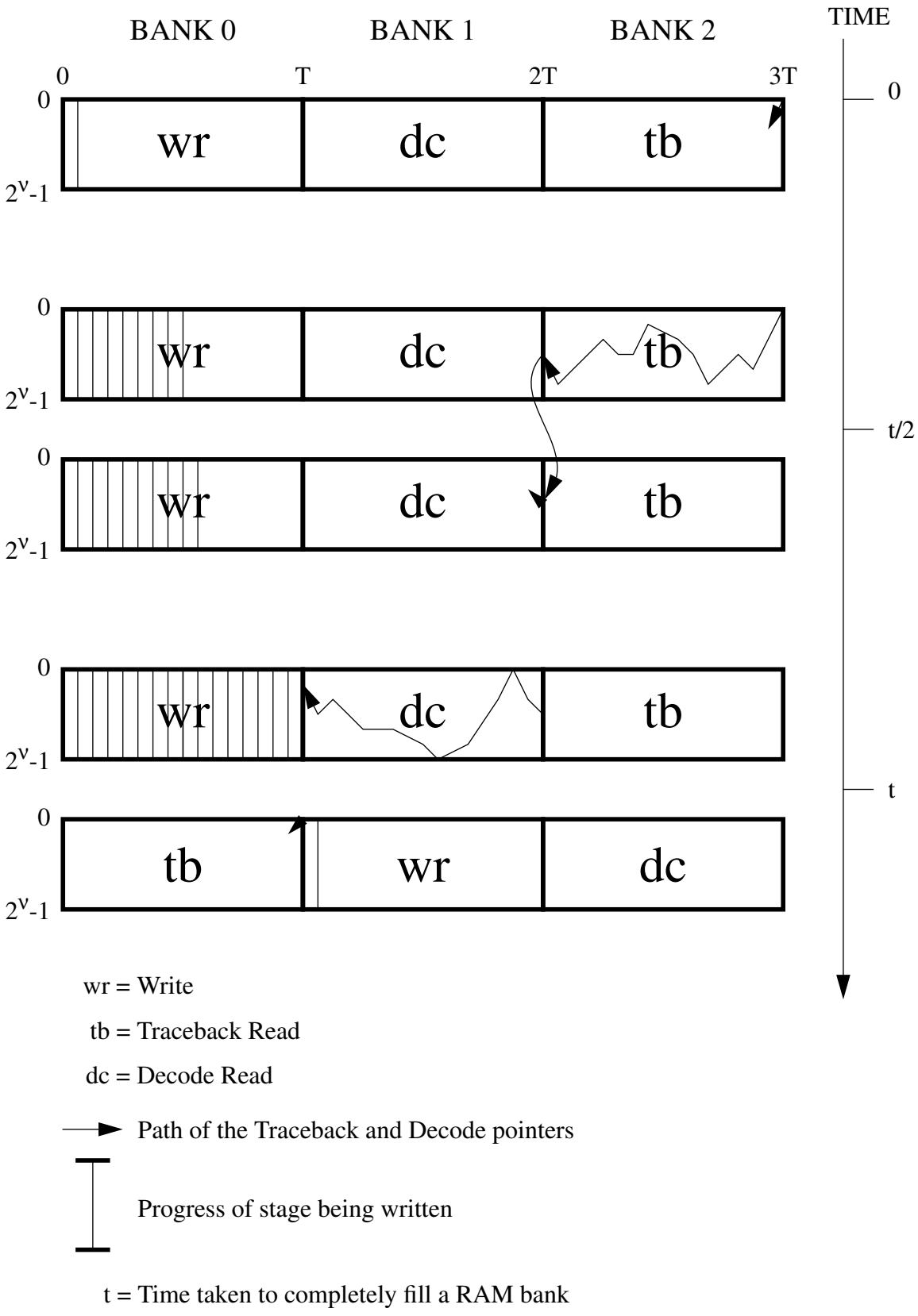



**Figure 4-14:** ***Two-pointer Traceback Example***

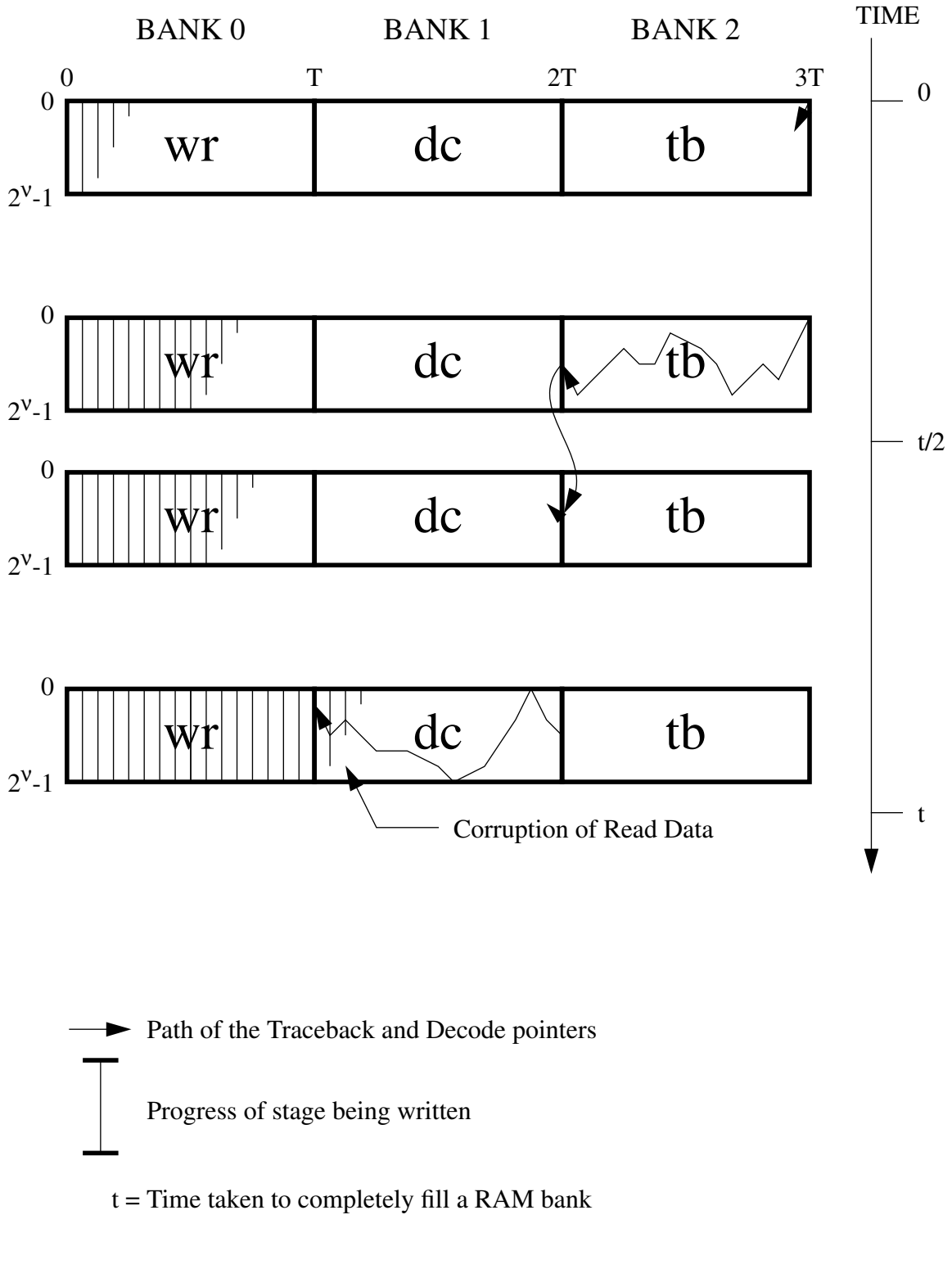



**Figure 4-15:** ***Example of Read/Write Pointer Collision***

The total number of bits required for buffering all stages would be:

$$\text{Total Bits} = 2^n \sum_{j=0}^{\nu-3} (t_{\nu-2} - tj) \qquad \text{(EQ. 4-2)}$$

Substituting EQ (3-9) and EQ (3-10) into EQ (4-2), the Total Bits simplifies to:

$$\text{Total Bits} = 2^n \left[ \frac{(\nu^2 - 3\nu + 2)\ (dpe + dsw)}{2} + (\nu - 3)\ (2^{\nu-n-1}) - n + 2 \right] \qquad \text{(EQ. 4-3)}$$

Using EQ (4-3), the total size of the buffer space required for the RACER is 92296 bits. This again exceeds the amount of memory available in the FPGAs. Since not all of the FPGA resources are used in the implementation of the memory controllers, a better method can be developed that does not require any extra external buffering.

Instead of having the read pointer move at exactly twice the speed of the write, we could make the read pointer move faster. This way, the interference can be avoided since the read pointer would complete the Decode Read before the write has any chance of overwriting the data. A drawback of this approach is that the read pointer must stay idle so that there is time for the RAM bank to complete writing. Having all writes complete at different times also poses a problem since ν-1 write addresses must be generated instead of one. Nevertheless, all the extra control logic can be easily implemented in the FPGAs used for the Traceback.

Another issue that must be resolved is the recovery of the decoded information. Recall from Section 3.2.1 that the ordering of the butterflies is a circular rotation of the butterfly number *j* positions to the right and that the output states determined are a 0 or 1 followed by the butterfly number. If the decisions are stored sequentially in memory, the state number and the memory address would not correspond. Referring to Figure 3-3, the decisions for PE0 could be stored as shown in Table 4-2.

| Memory Address | State |
|---|---|
| 0000 | 0000 |
| 0001 | 1000 |
| 0010 | 0100 |
| 0011 | 1100 |
| etc. | etc. |

**Table 4-2:** ***Sample Storage of States in Memory***

Two approaches can be made in the storage of the state values in memory. These are either to write the state decisions according to the state number, or to store the decisions sequentially and decode the read addresses to find the correct memory location. Since reads are less frequent than writes, more time can be given to decode the read address; as a result, decoding the addresses for the read pointer is the preferred choice. To decipher the correct memory location for a specific state, the lower ν-1 bits of the state number must be circularly rotated $j$ positions to the left to form the upper ν-1 bits of the deciphered address and the upper bit of the previous state, moved to the lower bit. For example, if the state number is represented by $\{S_{\nu-1}S_{\nu-2}...S_1S_0\}$, then the deciphered address is:

$$RAM\ Address = \{S_{\nu-j-2}S_{\nu-j-3}...S_1S_0S_{\nu-2}...S_{\nu-j-1}S_{\nu-1}\} \quad \text{(EQ. 4-4)}$$

Note that this requires little or no resources in hardware.

### 4.6.2 Traceback for the RACER

An integral part of the traceback is the memory configuration. From the previous discussion, recall that in the cascade architecture, decisions for the stages are written at different intervals of time. To allow the independent reads and writes, localized memory is used for each processor stage. Ideally, dual-port memory would be used for the simultaneous reads and writes; however, being costly, single-port RAMs are used instead. The problem with using single-port RAM is that only one address may be read or written at any given time. To emulate the dual-port memory, two clock cycles are used to interleave the read and write cycles which is double the

number of clock cycles for the dual-port memory. This also requires the decision datapath must be doubled, to achieve the same throughput as the dual-port RAM. Therefore instead of having four bits written in one cycle, a four to eight bit conversion is made and the eight bits are written every other cycle. In this way, reads may be performed between the write cycles.

For the RACER, the total amount of RAM required just for the Traceback Read is the number of states in one stage multiplied by the truncation depth (T). As a general rule, the truncation depth is chosen to be approximately $10 \times \nu$ for low SNR environments [Onys89]. Therefore, the amount of memory required to implement just the Traceback Read is $2^{\nu} \times (10 \times \nu) = 2^{14} \times 10 \times 14 = 2293760$ bits. If all three RAM banks are of equal size as proposed in Section 4.6.1, the total minimum memory required for the RACER would be $2293760 \times 3 = 6881280$ bits. However, by taking advantage of the faster read pointer, an alternate memory configuration can be derived. Recall that the reason for having equal sized memories was to allow the reads to proceed at exactly twice the speed of writes; thus, reducing the amount of control logic. However, the number of stages traversed by the Decode Read does not have to be the same as the Traceback Read. As long as the number of stages written is the same as the number of stages read by the Decode Read, the Traceback Algorithm is still valid.

If we now define a complete stage ($2^{14}$ decision bits) as a Virtual RAM Bank, then the previous configuration could be imagined as being partitioned into $10 \times \nu \times 3 = 420$ Virtual RAM Banks with only a third or 140 actually used for Traceback Read. The size of memory required for each processing stage just to implement the Traceback Read would be $\frac{2293760}{13} = 176444$ bits. Since the data bandwidth is eight bits and the smallest standard RAM size to hold the bits is 256 Kbits ($2^{18}$ bits), a 32K x 8 SRAM for each processing stage is sufficient. It can then be partitioned into $2^{18}/2^{14} = 16$ Virtual RAM Banks of memory per processing stage. For Traceback Read, the number of stages that must be traced by each processing stage is $\frac{10 \times \nu}{\nu - 1} = 10.77$. Since the size of the Decode Read and Write must be equal, each RAM can be partitioned to 12 Virtual RAM Banks for Traceback Read, two Virtual RAM Banks for Decode Read and two Virtual RAM Banks for Write. The timing for this final configuration is shown in Figure 4-16.

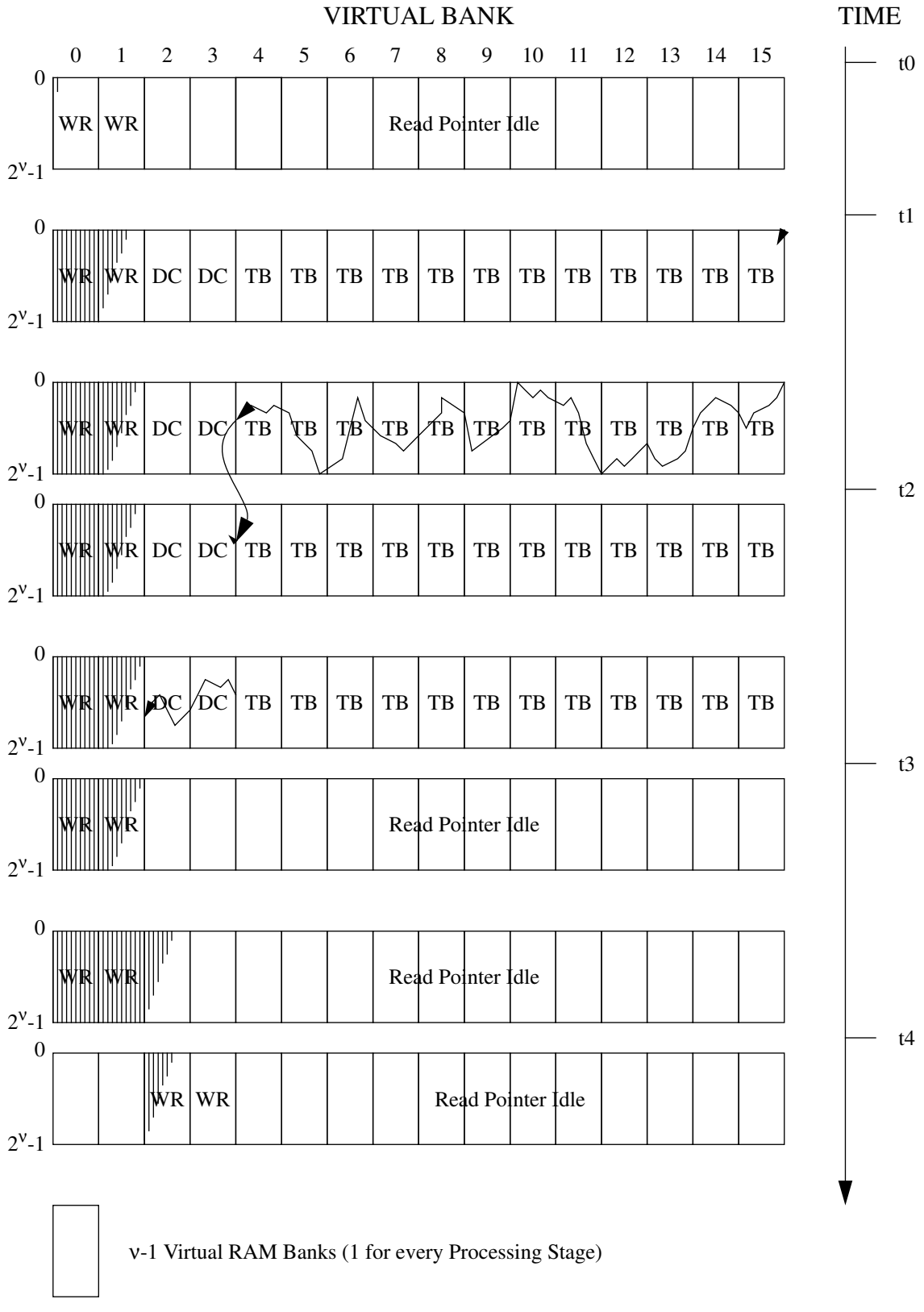



**Figure 4-16:** ***Timing for the RACER Traceback Algorithm***

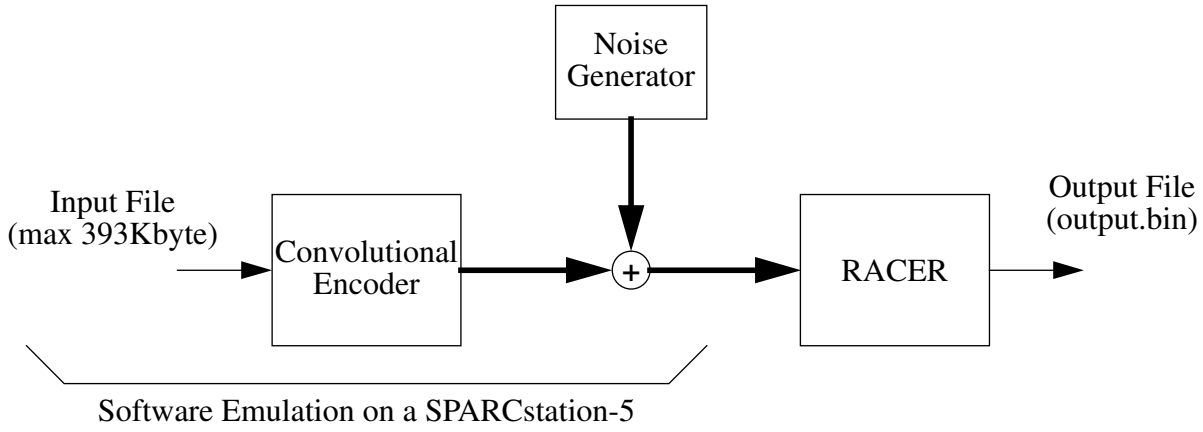



**Figure 4-17:** ***Experimental Setup***

## 4.7 Performance Data

The RACER has been exercised extensively with the help of a SPARCstation-5 emulating the convolutional encoder and noise generator. A block diagram of this setup is shown in Figure 4-17. Through these tests, we have determined that the peak throughput is approximately 41 Kbit/s with a 16 MHz system clock. However, from EQ (3-14), the theoretical data rate should have been:

$$\text{Data Rate} = \frac{13}{[(13)(5+2) + 2^{14-2} - 1] \times (16\times10^{6})^{-1}}$$
$$= 49677 \text{ bits/s}$$

which is about 20% higher. The reason for the discrepancy is the fact that the SUN interface cannot sustain the 49.6 Kbit/s data rate. As a result, the RACER must occasionally stall to allow time for the input buffer to fill up. If the input buffer is empty or 1/16 full, the RACER will stop, allowing the buffer to fill up to 1/4 full. Once half full, the RACER is permitted to continue until the buffer is almost empty and the cycle is repeated. Therefore, during the time when the buffer is not empty, the RACER will be running at full speed.

Table 4-3 shows a comparison of the various implementations of the Viterbi decoder. Estimating the actual cost of a system is extremely difficult. The cost comparisons shown in Table 4-3 are a little misleading since the cost of the Uni-Processor, DECPeRLe, RACER and RACER II only take into account the price of purchasing either the system or the materials needed to build

the system. On the other hand, the BVD takes into account materials cost, labour, research and development over a span of many years. Nevertheless, rough comparisons can be made. It can be seen that the RACER is the second cheapest and second only to the BVD in throughput. The price per decoded bit/s for the RACER and BVD are 30000/49600 = $0.60/bit/s and 2000000[1]/1000000 = $2.00/bit/s respectively. Compared to the BVD, the RACER is 3.3 times cheaper for every bit/s but at 1/20 the decoding rate. This does not indicate an advantage towards using the cascade approach, however, Section 5.2 describes a 1 Mbit/s decoder (RACER II) using FPGAs that would still be cheaper than the BVD.

An indication of the wiring complexity of the BVD is shown by the number of layers required for its printed circuit board (PCB) backplane, which is 28 compared to only six for the RACER.

| | **Cost ($ Cdn)** | **Decode Rate (bit/s)** | **#Chips for ACS** | **# of Layers for PCB** | **Clock (MHz)** |
|---|---|---|---|---|---|
| Uni-Processor (SPARCstation-5) | < $10,000 | < 50 | 1 microSPARC II | N/A | 70 |
| BVD Prototype | Multi-million dollar project[a] | 1,000 | N/A | 2 Large Racks of PCBs | 20 |
| BVD Prototype II | | 1,000,000 | 256 custom VLSI chips | 28 | 25 |
| DECPeRLe-1 | ~ $40,000 | 2,000 | 16 XC3090 FPGAs | N/A | 25 |
| RACER | ~ $30,000 | > 41,000 (Theoretic 49,600) | 26 XC4010 FPGAs | 6 | 16 |
| RACER II | ~200,000 (estimated) | 1,000,000 | 104 XC4025 FPGAs | 6 | 48 |

**Table 4-3:** ***Comparisons of Viterbi decoders based on the BVD specifications***

a. From informal conversations with people in the project.

1. Do not really know actual cost but we understand that it is in this range.

Experiments involving the Bit Error Rates for various Signal to Noise Ratios have also been performed. The software used to generate the Gaussian noise was provided by JPL for their simulations of the Viterbi decoder. Figure 4-18 shows the results of the experiment, which is almost identical to the BVD [Stat88]. The results for the BVD are not shown for clarity since they essentially overlap the curves shown.

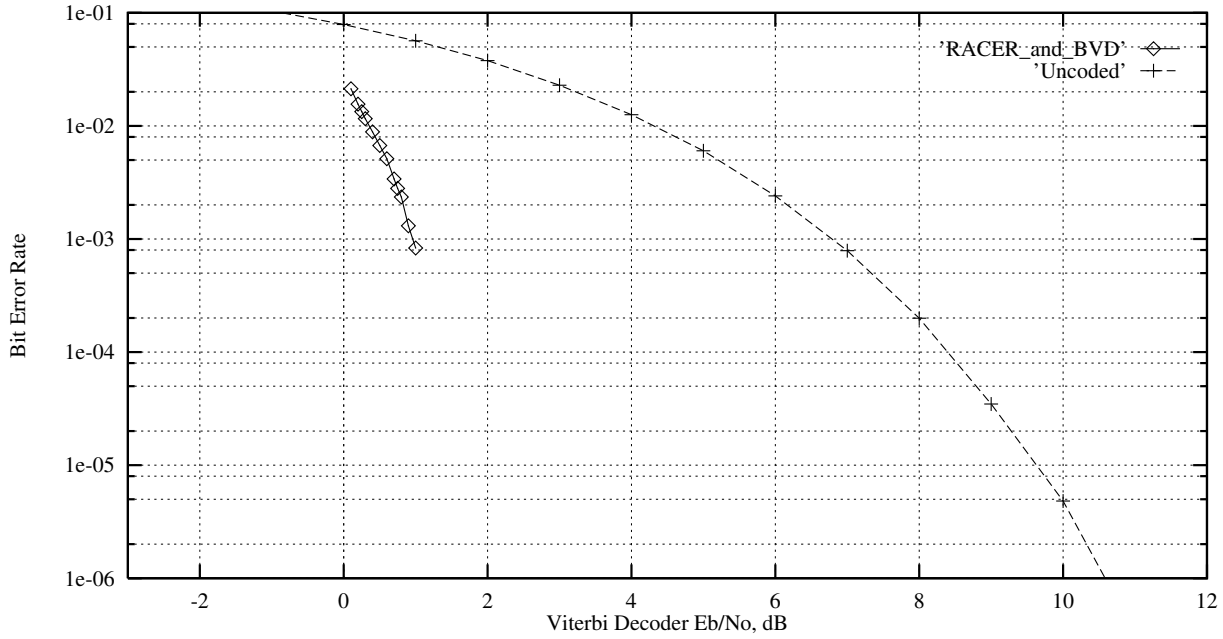



**Figure 4-18:** ***Performance Comparison for the RACER and BVD***

## 4.8 Summary

In this chapter, a summary of the RACER layout was provided. A method was described to allow the design of one daughter board for many different configurations. We showed how proper floorplanning would lead to denser designs in FPGAs. The timing of the cross-point switches were presented. We explained the two-pointer traceback and showed how the survivor management memory was divided for the RACER. Finally, we demonstrated that the RACER has the same functionality as the BVD.

*Chapter* **5**

# Multi-Ring Viterbi Decoders at 1 Mbit/s and Beyond

## 5.1 Introduction

Now that the multi-ring GCVD architecture, with functionality comparable to the BVD, has been proven successful in hardware, the next step is to achieve higher decoding rates. The fastest Viterbi decoder to date with the same specifications as the RACER is the second prototype of the BVD [Onys91] with a decoding rate of 1 Mbit/s. Therefore, the next goal would be a 1 Mbit/s FPGA version of the RACER, which shall be called the RACER II. Once the final prototypes are complete, a faster 6.47 Mbit/s custom chip version called the RACER III could be considered.

## 5.2 The 1 Mbit/s RACER II

To obtain a throughput of 1 Mbit/s, the RACER II must be at least $\frac{1\times10^6}{50\times10^3} = 20$ times faster than the RACER. From EQ (3-14), it is apparent that for a fixed constraint length, the Data Rate may be improved by increasing the clock frequency of the system, by adding more rings and to a very small extent, by decreasing the number of pipeline stages. With recent advances in VLSI technology, it is not unreasonable for chips to be operating at hundreds of megahertz. Therefore, without any modifications to the RACER, full custom chips operating at approximately 322 MHz can be designed to replace all the FPGAs; however, the board itself must operate at that speed and the chips must be deeply pipelined. A drawback is that memory access times must be faster than

$322/2 = 161$ MHz if no further serial to parallel conversions of the decision bits are made. In the other extreme, a $2^{13}$ ring Viterbi decoder can be constructed with a system frequency of 16 MHz and a pipeline depth of less than 16. Obviously, this is impractical since it would require $2^{\nu-1} \times (\nu - 1) = 106496$ processing elements which is $\nu - 1 = 13$ times more than the fully parallel BVD architecture. In addition, there would be inefficient use of processor resources since twelve out of every thirteen processing stages will be idle. The final alternative is to develop a hybrid of the two extremes. The following sections will discuss the feasibility of building such a board level multi-FPGA Viterbi decoder and propose an architecture.

### 5.2.1 Requirements

A consideration to building the new Viterbi decoder is the amount of hardware required. For a board level design, a reasonable system speed would be about 50 MHz. Using this specification and assuming that the number of pipeline stages is equivalent to the RACER ($dpe + dsw = 7$), the minimum number of rings can be calculated from EQ (3-14).

$$1 \times 10^6 = \frac{(14-1) \times 50 \times 10^6}{[\,(14-1)\,(7) + 2^{14-n} - 1]}$$

$$n \sim 5$$

Therefore, a $2^{n-1} = 2^4$ ring Viterbi decoder is sufficient for a 1 Mbit/s decoding rate. With this architecture, the minimum system speed is 46.4 MHz. However, is a $2^4$ ring Viterbi decoder difficult to route as in the fully parallel Viterbi decoder? Moreover, is the present type of hardware sufficient for such a design? The following sections discuss these problems.

### 5.2.2 Routing

Routing for any large system is complicated. For the fully parallel architecture, routing is difficult due to the substantial number of crossovers required to connect all the processors. To route such a board would require many signal layers and vias. On the other hand, the multi-ring GCVD provides a simpler architecture. Data is passed locally from processor to processor via switches. However, as more rings are added to the GCVD, more processors are wired directly. Is there a routing problem similar to that of the parallel architecture? Figure 5-1 shows a $2^3$ ring, constraint length 4 Viterbi decoder that is unwrapped to show routing. Since $n = \nu - 1$, all processors are connected together directly without the use of any cross-point switches. It can be

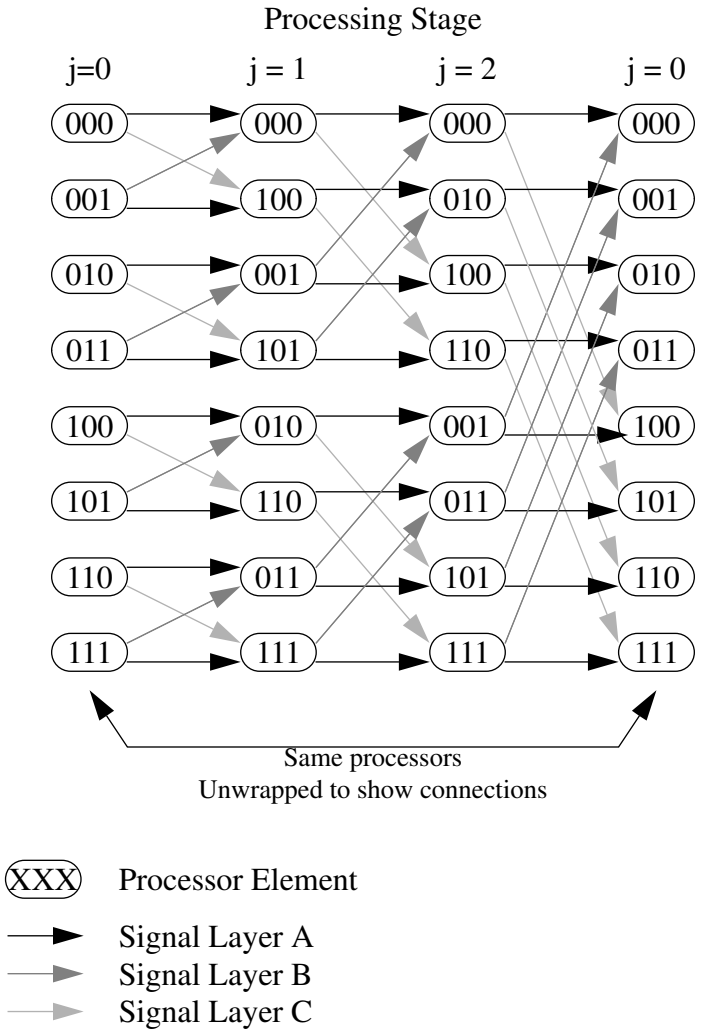



**Figure 5-1:** ***Wiring for a $2^3$ ring Viterbi decoder***

seen that efficient routing can be achieved with only three or six signal layers depending on whether the processing stages are arranged in a ring or interleaved as in the RACER. Three layers for signals going from the left to right direction and if the processing stages are interleaved, another three for the opposite direction to wrap around and complete the ring. Specifically, these six layers are broken down to one layer for each of the horizontal left-right, diagonal up-right, diagonal down-right, horizontal right-left, diagonal up-left and diagonal down-left directions. Therefore, the first n routes for any $2^{n-1}$ ring Viterbi decoder can be efficiently connected with only six signal layers or fewer. A difficulty with this approach is the length of the traces. As the number rings increase, the length of the traces increase exponentially. However, as the number of rings increase, the system speed of the Viterbi decoder can decrease to achieve the equivalent throughput; therefore, the speed may be slow enough such that there is enough time for signals to

travel long distances. If there is insufficient time for signals to propagate, high speed drivers can be implemented at the output of the processor chips to drive the longest traces.

### 5.2.3 Processors

An important portion of the Viterbi decoder is the processor itself. The RACER presently uses 13 XC4010 FPGAs to implement the processors. If the same FPGAs are used, a total of $13 \times \frac{2^4 \text{ ring}}{2^1 \text{ ring}} = 104$ FPGAs would be required. This number, while large, is certainly fewer than the 256 custom ASICs used for the BVD. However, by using the denser XC4025 FPGAs [Xlxc93], even fewer chips would be required. Table 5-1 shows a comparison of two types of

| Device | XC4010 | XC4025 |
|---|---|---|
| Number of CLBs | 400 | 1,024 |
| Number of Flip-Flops | 1,120 | 2,560 |
| Max RAM Bits | 12,800 | 32,768 |
| Number of IOBs | 160 | 256 |

**Table 5-1:** ***Comparison between the XC4010 and XC4025***

FPGAs. The XC4025 has more than double the amount of most resources, and 1.6 times more input/output pins (I/O). As a result, four processing elements can be implemented in one XC4025, reducing the number of FPGAs to 52.

The I/O count for the XC4025 is of concern, since it does not have double the I/O of the XC4010. To implement the four processing elements, eight additional sets of 14-bit pins are required for the path metric data and another four one-bit pins are needed for the decision bits. As a result, an additional $8 \times 14 + 4 = 116$ pins are necessary. With 156 pins used for the present processor implementation, the total number of pins becomes 272 which is 16 more than the number available on the XC4025. To reduce the number of I/O, some data must be multiplexed. For the RACER, the sample data is the ideal choice for multiplexing since there is a minimum of seven clock cycles between samples. Referring to Figure 5-2, four clock cycles can be used to load the sample data. By using this method, the sample data bandwidth can be reduced from

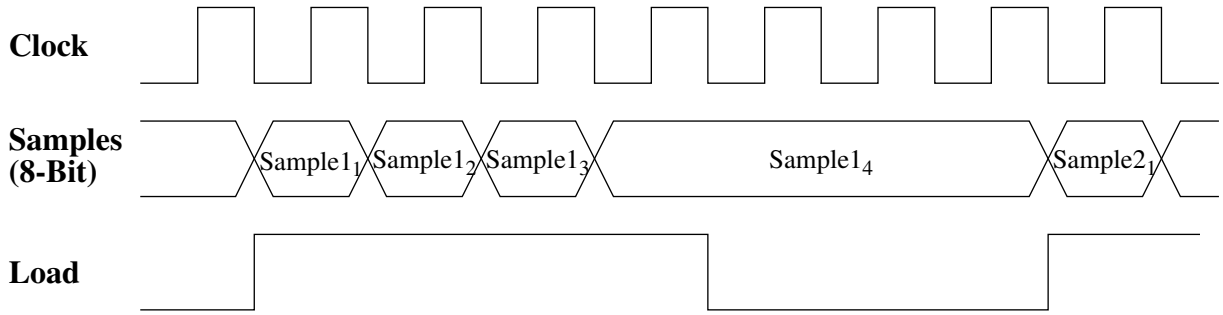



**Figure 5-2:** ***Timing to Multiplex the Loading of the Sample Values***

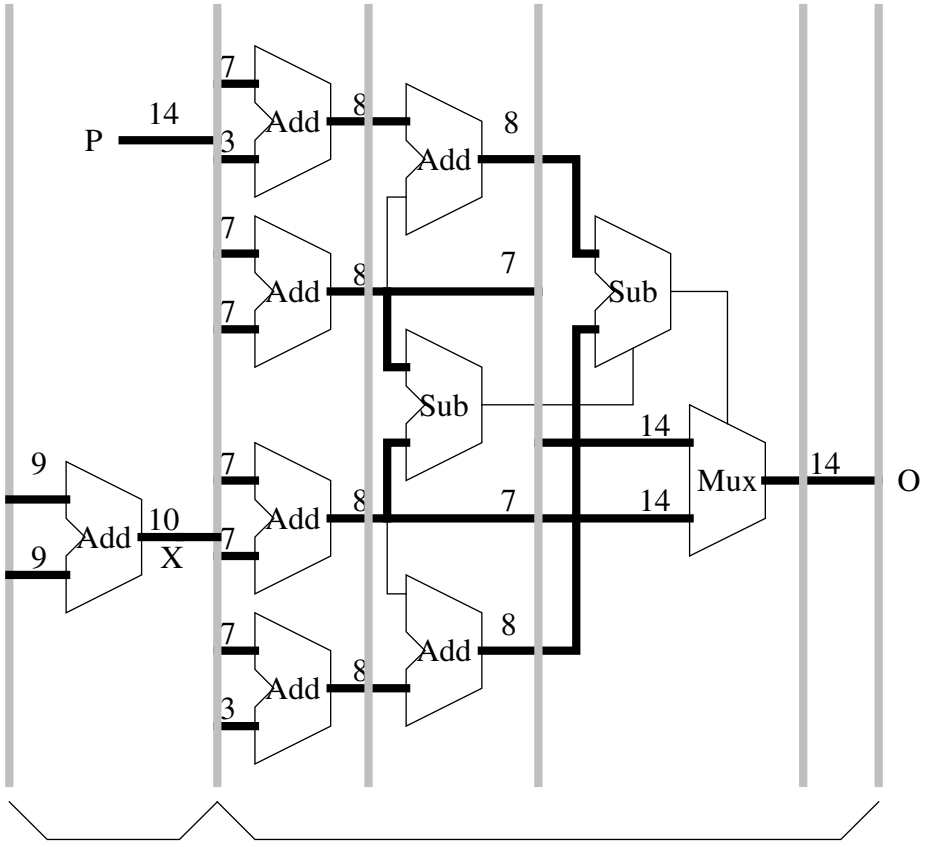
$4 \times 8 = 32$ to eight pins. Therefore, the total number of I/O required is reduced to 248 which is less than 256 available on the XC4025.

Again speed is the limiting factor in the implementation of the processor chip. The FPGAs used for the RACER are of grade '-5'; however, at the time of this writing, a '-2' grade is available which is 2.5 times faster. A simple replacement of the '-5' FPGAs with the '-2' would increase the operating speed from 16MHz to 40MHz. This is a little less than the required 46.4 MHz; nevertheless, it might be possible to pipeline the processor even further to increase the operating speed. Referring to Figure 4-8, two additional pipeline stages are necessary to reduce the size of the adders to 5-bits or less. The first pipeline stage is to break the 9-bit adder in the BMG to a 5-bit adder and a 4-bit adder. Secondly, instead of breaking up the 14-bit adders in the ACS to two 7-bit adders, 5-bit and 4-bit adders will be used. From experimentation, the 5-bit adder is 1.2 times faster than the 7-bit adders. Therefore, if pipelining is possible, the operating speed would increase to $40 \times 1.2 = 48$ MHz which is enough to achieve the 1 Mbit/s data rate.

From Table 4-1, approximately 85% of the XC4010 is actually used. Therefore, the number of CLBs free in the XC4025 if double the processor logic in the XC4010 is implemented is $1024 - (400 \times 0.85 \times 2) = 344$. However, the number of CLBs can be reduced since there is some redundancy in implementing two XC4010s. The first redundancy is the memory used to store the generator polynomials and the counters. If the second set is removed, 40 CLBs would be saved. Secondly, referring to Figure 4-8, only one set of the two eight bit adders in the BMG are required; therefore, the adders from the seven other PEs can be removed saving

#CLBs in 9-Bit Adder $\times$ #Adder $\times$ #Redundant PE $= 6 \times 2 \times 7 = 84$ CLBs. Finally, another 24 CLBs can be saved by removing the logic in the second set of parity generators. Therefore, a maximum of $344 + 40 + 84 + 24 = 492$ CLBs can be used to implement the new pipeline stages.

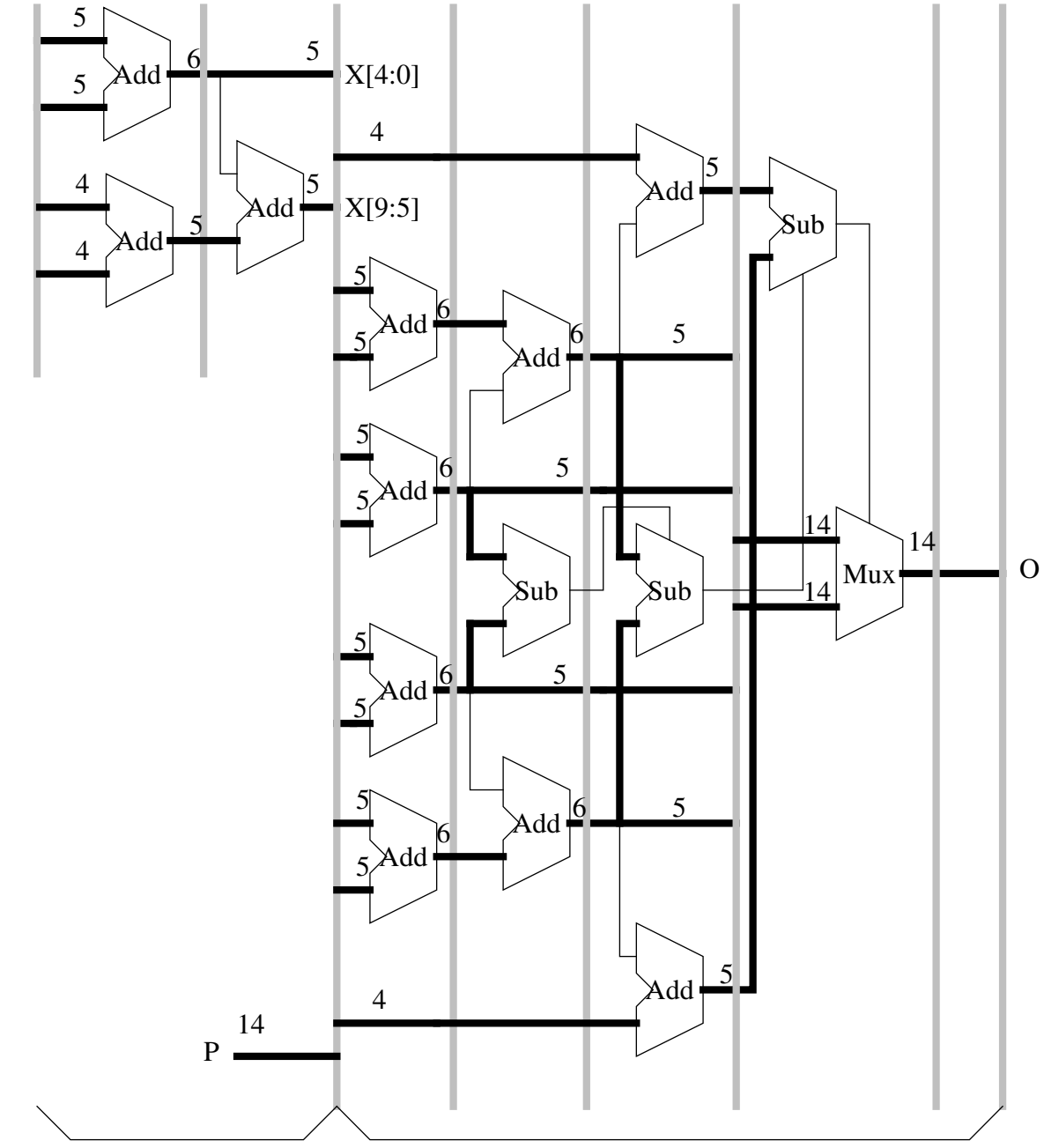
The pipelining for half of one PE might look like Figure 5-3. Assuming that the 9-bit, 8-bit, 7-bit, 6-bit, and 5-bit, adders/subtracters with flip-flops take 6, 5, 5, 4, and 4 CLBs respectively and that every 2 flip-flops take one CLB, then the total number of CLBs in the new design is 96. From Figure 5-4, half a PE in the RACER would require 69 CLBs. As a result, 27




**#CLBs = 13 + 20 + 23 + 13 = 69**

X = #Bits in Datapath

**Figure 5-4:** ***Pipelining for half a PE in the RACER***




**#CLBs = 8 + 14 + 20 + 21 + 22 + 11 = 96**

X = # Bits in Datapath

**Figure 5-3:** ***Possible new pipelining for half a PE in the RACER II***

additional CLBs are needed to implement the extra pipelining for half a PE. With eight PEs in the XC4025, this translates to an extra 432 CLBs which is less than the 492 CLBs available. The %utilization of the XC4025 is $\frac{1024 - 492 + 432}{1024} \times 100 = 94.1\%$, which is extremely high but with very careful floorplanning, it should be possible because of the regularity of this design.

### 5.2.4 Cross-Point Switches

As with the processors, the XC4025 have sufficient resources to implement four cross-point switches. For the extra two cross-point switches, additional sets of eight 14-bit pins are required for a total of 112 pins. With the 115 pins presently used for the RACER, the XC4025 has more than enough I/O.

An advantage to using more rings is that fewer FIFOs are required per cross-point switch at each processing stage. For a single-ring GCVD, the deepest FIFO would be $2^{\nu-2}$. Therefore, for a $2^4$ ring GCVD, the deepest FIFO would be $2^{\nu-2}/2^4 = 256$. With four cross-point switches for every FPGA, and two 14-bit wide FIFOs for every cross-point switch, the total size of the FIFO would be $256 \times 4 \times 2 \times 14 = 28672$ bits. Including the necessary combinational logic, the XC4025 lacks sufficient resources to implement everything. However, recall from Section 4.3 that the four 128Kx14 FIFOs were successfully implemented in a XC4010; therefore, for the RACER II, four or more 256Kx14 FIFOs can be implemented in the FPGA. The remaining FIFOs can either be implemented with four or less external 256Kx14 FIFOs, or if the backplane-daughter board architecture similar to the RACER is adopted for the RACER II, the remaining FIFOs can be implemented in the dummy FPGAs on the last daughter board. This is possible since there are 14 FPGAs available to implement the processors on the RACER but only 13 are actually used.

### 5.2.5 Memory

Fortunately, the total memory required for Traceback in the RACER II is the same as the RACER. What does change is the memory bandwidth does change. With 16 processing elements simultaneously providing two decisions each, the data bandwidth of the memory must be 32 bits. Recall from Section 4.6.2 that a 256K RAM is required at each processing stage for Traceback. For dual-port RAMs, any configuration leading up to 8Kx32 is suitable. If this configuration is used, then the memory controllers would require an extra $32 \times 2 + 24 \times 2 = 102$ pins for I/O and

$13 \times 2 - (15 - 2) \times 2 = 22$ pins for memory addresses. Adding this to the 97 pins already used for the RACER, the total becomes 221, which is less than the number of pins in the XC4025. Moreover, for the RACER II running at 48.4 MHz, the RAMs must have access times of less than 20 ns which is already commercially available.

On the other hand, a 4Kx64 single-port RAM would be needed if the configuration similar to RACER is implemented. An additional $32 + 24 \times 2 = 80$ pins are needed for I/O and 4 fewer for the memory addresses. Again, the XC4025 has more than sufficient I/O. Memory access times of less than 40 ns is also necessary.

Either memory configuration described above could be easily implemented. The deciding point would be cost, since dual-port RAMs are more expensive than singe-port RAMs.

### 5.2.6 Layout

Now that all difficulties with the FPGAs and memory have been resolved, the next step is determining an appropriate layout. Attempting to construct a single board RACER II would be very difficult since there are over a hundred XC4025 FPGAs. If a grid of 10x10 FPGAs are laid out, and if each FPGA is 3 cm x 3 cm, then the minimum dimensions for board would be 30 cm x 30 cm. This configuration does not account for the other components and the routing between components. Moreover, as in the RACER, testing would be difficult. Therefore, a backplane-daughter board arrangement similar to that of the RACER is recommended.

What type of configuration is suitable? The best module would consist of the components necessary to compute one stage of the trellis. Specifically, four FPGAs for the processors, four FPGAs for the switches, one FPGA for traceback, and RAM. If this module is used, 13 daughter boards are required. This method has a drawback since the backplane must be larger to accommodate the daughter boards. Moreover, additional FPGAs are required to implement the memory controllers. A better method is to implement two of the modules on a daughter board as in the RACER. This way, only seven RAM controllers would be required since only seven daughter board are needed. In addition, with seven daughter boards, a constraint length 15 Viterbi decoder can be constructed. The drawback however is that the area almost doubles for a single sided board.

The other alternative, is a two-sided board that has two modules. There are many advantages to this approach. Firstly, the size of the board would be close to the one module single-sided board. Furthermore, recall that for the $2^4$ ring GCVD, the first four connections between processing stages are direct. Since the first two hardwired connections can be easily routed in the switches, the layout of the board would be identical to the boards with FIFOs. Therefore for a single-sided board design, three different types of custom daughter boards would be required: two for the last two hardwire connections and one for the general daughter board. On the other hand, only two custom boards would be needed if the two sided board is implemented. The diagrams for the two daughter boards are shown in Figure 5-5 and Figure 5-6. Note that the first layout only requires two signal layers and as discussed in Section 5.2.2, the second layout requires at most six signal layers.

If it is not necessary for the RACER II to be reconfigurable as a constraint length 15 Viterbi decoder, then a better alternative to the above can be devised such that there would be only one type of daughter board. Referring to Figure 5-6, the reason for having the second type of daughter board was to allow for the hardwired routing between the processor and switch. If the hardwire routing could be done in the backplane, then the extra daughter board would not be required. Referring to Figure 5-7, each daughter board has now been partitioned into SW-PE-SW-PE as opposed to PE-SW-PE-SW. Recall that of the seven double sided daughter boards built, one of the daughter boards will have an unused side since only 13 of the 14 processing stages are actually used to implement the constraint length 14 Viterbi decoder. Therefore, SW2 and PE3 could be placed alone on this special board leaving the second side with the locations for the PE and SW shorted. With this configuration, the hardwired portions can be routed in the backplane. The final daughter board is shown in Figure 5-8. There is another drawback to this approach. Recall in Section 5.2.4, that it may be necessary to use the second side of the last daughter board to implement the remaining 256Kx14 FIFOs. Since the second side is no longer available, external FIFOs must be used to implement the 256Kx14 FIFOs.

The layout of the backplane is identical to the RACER (Section 4.3) where the daughter boards are interleaved to achieve the same controlled distance between boards (Figure 4-2).

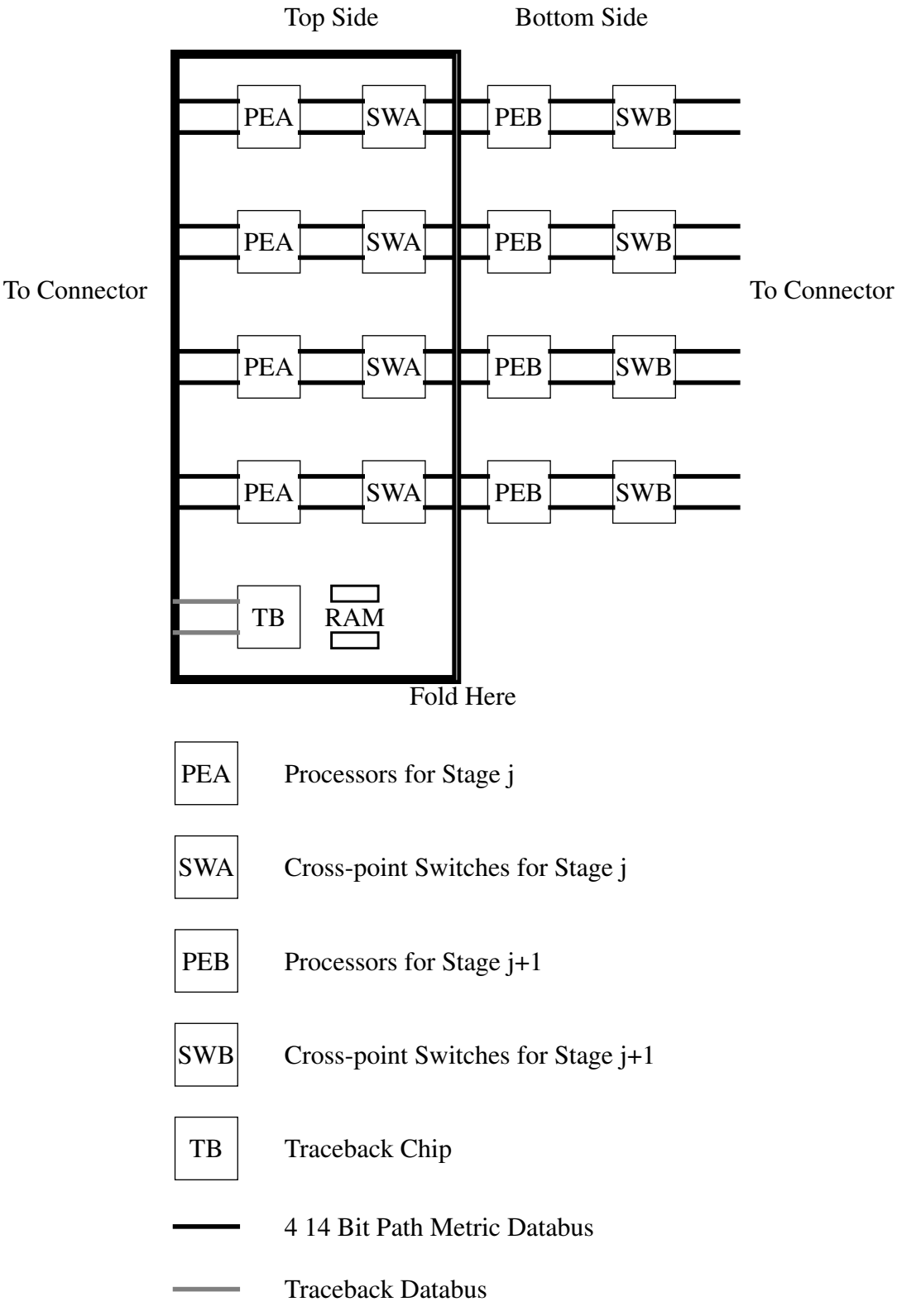
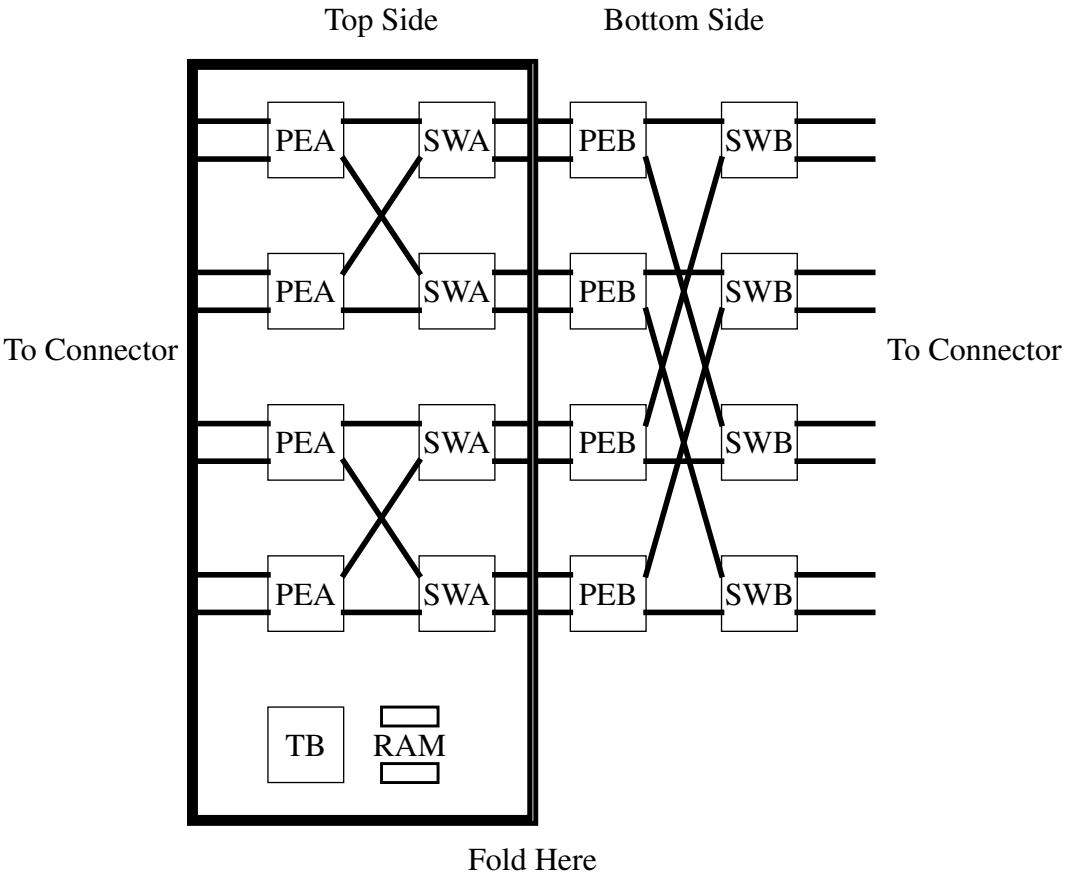



**Figure 5-5:** ***Layout for Daughter Boards 1 and 3 to 7***



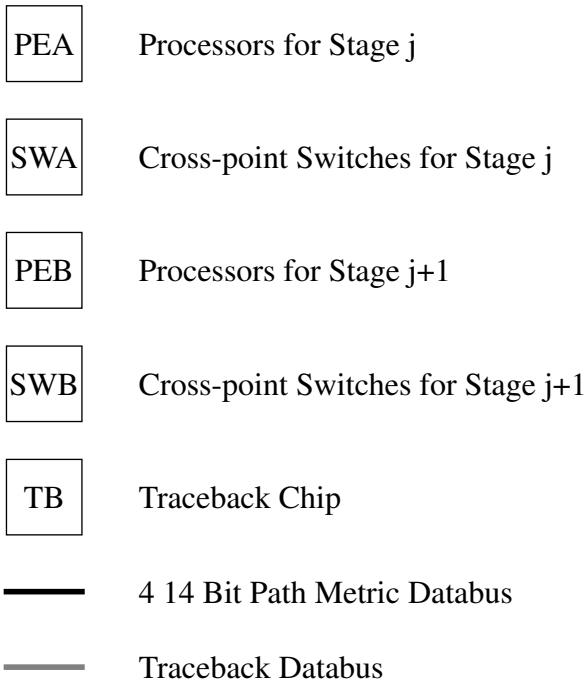




**Figure 5-6:** ***Layout for Daughter Board 2***

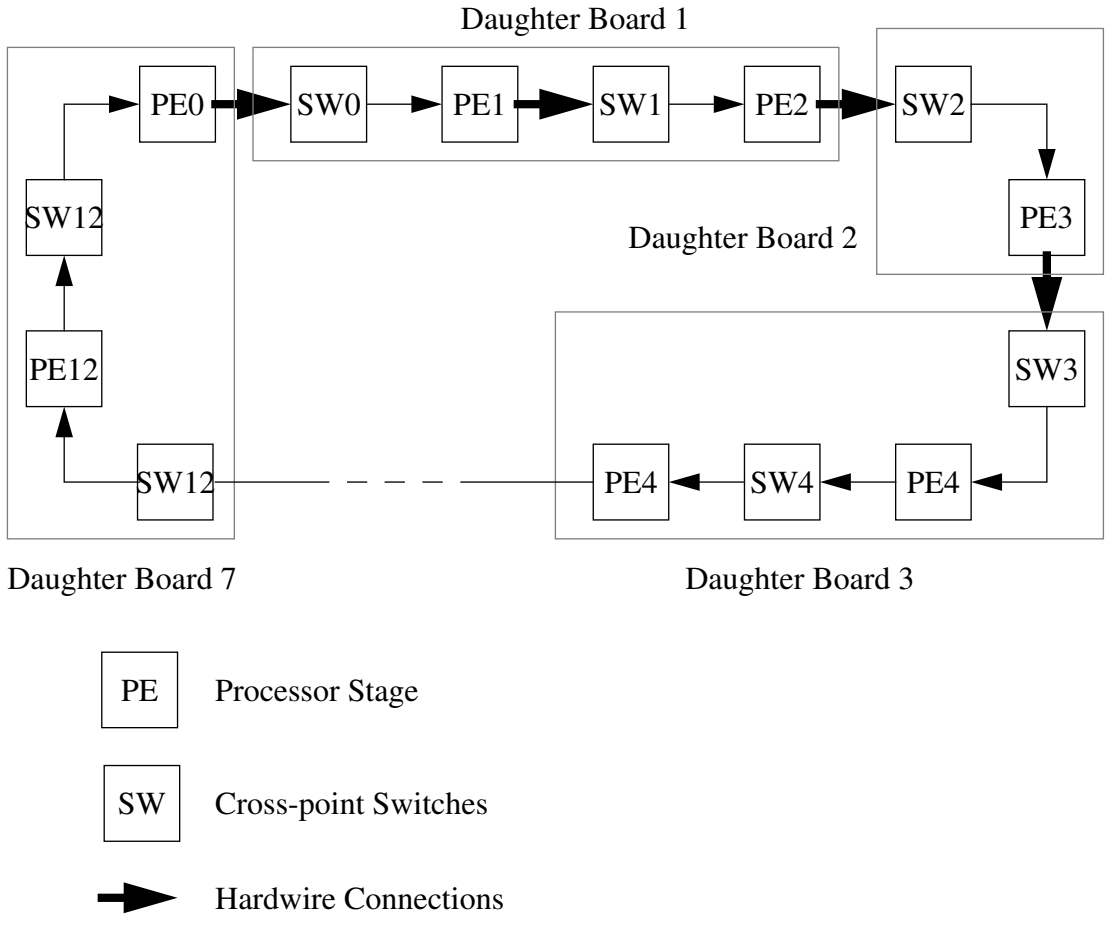
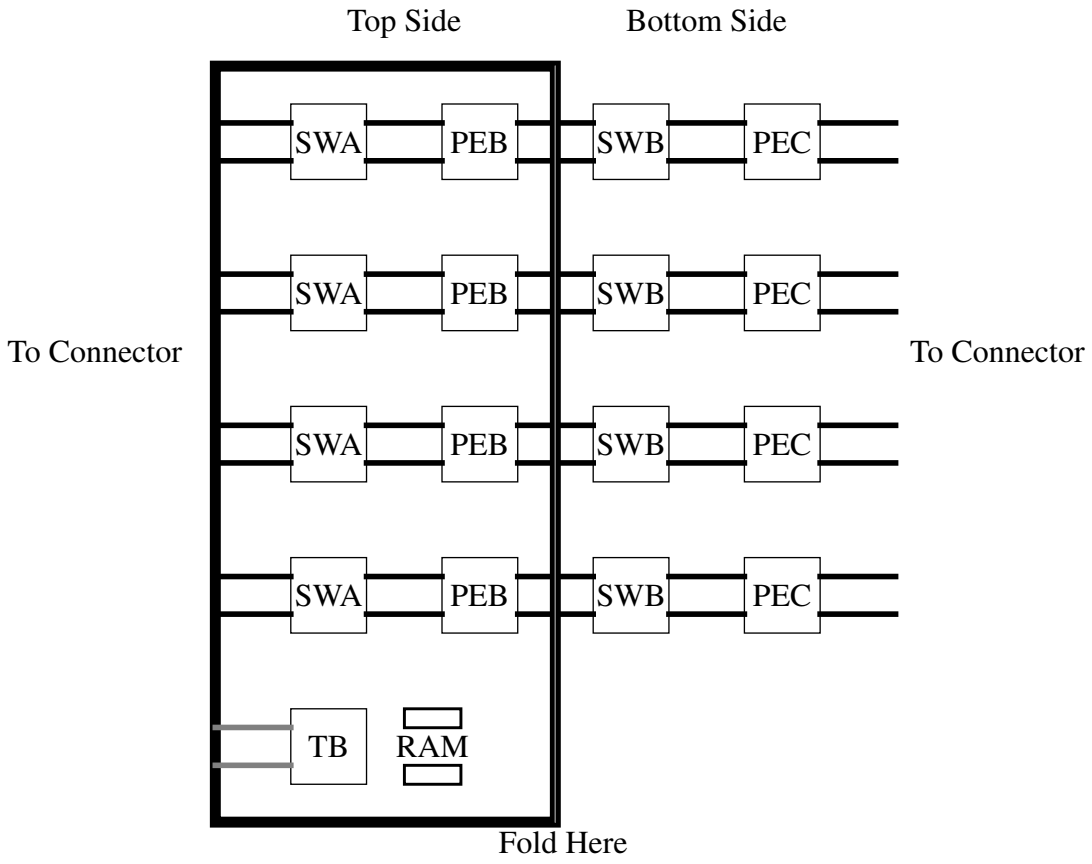



**Figure 5-7:** ***Alternate Configuration for the RACER***

### 5.2.7 Estimated Cost

A rough estimate of the total cost of the RACER II can be determined. Assuming the cost of the new two sided boards, labour and other components amount to the total cost of the RACER, then the only cost to add is the price of the XC4025, which is approximately $1500 each. With 52 FPGAs for the processors, 52 FPGAs for the switches, seven FPGAs for the memory controllers and one FPGA to control the whole system, a total of 112 FPGAs are required. In total, the cost of the RACER II is approximately $30000 + (1500 \times 112) \approx \$200000$, which is 10 times cheaper than the BVD at the same decoding rate and without the need for custom ASICs.




SWA — Cross-point Switches for Stage j mod (ν-1)

PEB — Processors for Stage (j+1) mod (ν-1)

SWB — Cross-point Switches for Stage (j+1) mod (ν-1)

PEC — Processors for Stage (j+2) mod (ν-1)

TB — Traceback Chip

4 14 Bit Path Metric Databus

Traceback Databus

**Figure 5-8:** ***Daughter board for SW-PE-SW-PE Configuration***

## 5.3 The RACER III

The main limitations for the RACER and RACER II are the maximum speed of both the FPGAs and the board. What if it were possible to fit the multi-ring GCVD onto a custom chip? With this type of implementation the system speed is no longer limited by the board. Moreover, customizing the logic within the chip allows for faster speeds. At the time of this writing, there exists 300 MHz DEC Alpha 64-bit RISC Microprocessors designed with a 4-layer metal, 0.5 μm CMOS process [Bowh95]. If this technology were available, implementation of a large multi-ring, single-chip Viterbi decoder would be possible. With the same architecture as the RACER II, a 300 MHz chip version which we will call the RACER III could achieve a data rate of:

$$\text{Data Rate} = \frac{(14-1) \times 300 \times 10^6}{[\,(14-1)\,(7) + 2^{14-5} - 1\,]}$$
$$= 6.47 \text{ Mbit/s}$$

A drawback to this approach is reconfigurability. The RACER III is no longer as flexible as the RACER II since most of the logic is fixed. Nevertheless, the concept of the RACER III is for a realization of a commercial chip. Therefore, the next few sections will discuss the implementation of a 6.47 Mbit/s RACER III.

### 5.3.1 Input/Output Requirements

Fortunately, the RACER III does not require many signal I/O, since, the path metric and memory data lines do not come off chip. The only I/O required are those for the input samples, the decoded output, control and testing. With 32 pins for input, one pin for the decode output, and approximately 20 for control and testing, the total I/O would be 53.

If the technology did not permit the traceback memory to be implemented on-chip, then the RACER III would not be possible. Recall that for a 16 ring GCVD, there are 32 decision bits per stage and with 13 stages a total of 416 pins would be required. More pins would actually be necessary since the decision bits must be buffered to reduce the rate at which decisions come off chip. For example, assuming that the board containing the RACER III and the external memory operates at 100 MHz, and the RACER III runs at 300 MHz. After every three cycles of the RACER III, one memory access can occur. Therefore, a 1:3 buffer must be provided to store the

decision bits of the three cycles. As a result, $416 \times 3 = 1248$ pins are required, which is impossible with today's technology.

### 5.3.2 Die Size

An important factor in the implementation of the RACER III is its size. In the RACER III, the largest contributions in area come from the core logic used to calculate the state path metrics and the memory used to store the decision bits. We shall use the DEC Alpha as a benchmark to the possibility of implementing the RACER III onto a chip. The DEC Alpha is implemented on a 16.5 mm x 18.1 mm die with 4 layers of metal and a 0.5 µm CMOS process.

The amount of memory required for traceback can be determined from Section 4.6.2. Recall that 10.77 Virtual Banks are necessary for the Traceback Read for each processing stage. Therefore, using a survivor memory configuration of 11 Virtual Banks for Traceback Read, one Virtual Bank for Decode Read and one Virtual Bank for Write, the total memory used for the survivor manager is $2^{\nu} \times (11 + 1 + 1) \times (\nu - 1) = 2768896$ bits or 346Kbytes. Memory for the cross-point switches amounts to $\left( \sum_{j=0}^{\nu - n - 1} 2^j \right) \times 2^{n-1} \times 14 \times 2 = 228928$ bits or 28.6 Kbytes. With the addition of approximately 1 Kbyte for the input and output FIFOs, the total memory required for the RACER II is $346 + 28.6 + 1 = 375.6$ Kbytes.

The actual die size for the memory can be estimated from the cache on the DEC Alpha. Using its 96 Kbyte cache, which is approximately 76 $mm^2$ from measuring the die photo, as a guideline, the memory area required for the RACER is $(375.6/96) \times 76 \approx 300$ $mm^2$.

The core logic size can be estimated from the constraint length 4 Viterbi decoder constructed at the University of Toronto using a 1.2 µm CMOS process [Feyg93]. In the University of Toronto Viterbi Decoder, a cell containing a processing element with its associated switch is approximately 0.763 $mm^2$. Scaling the cell down to a 0.5µm CMOS process and assuming that the RACER III requires twice the bandwidth, the new size of the cell is $0.763 \times 2 \times (0.5^2/1.2^2) = 0.265$ $mm^2$. With 16 processing elements per stage and 13 stages, the total area becomes $16 \times 13 \times 0.265 \approx 56$ $mm^2$. Since 26 out of a total of 34 FPGAs are used for the path metric calculations in the RACER, the total logic size is estimated to be $(34/26) \times 56 \approx 74$ $mm^2$.

Therefore, the total area of the RACER III chip approximately $300 + 74 = 374$ mm$^2$ which is about 1.25 times larger than the DEC Alpha. This is borderline for a single chip. A more reasonable size will be realized using better technology such as a 0.35 µm CMOS process which would reduce the die area to about $374 \times (0.35^2/0.5^2) \approx 184$ mm$^2$.

A possible floorplan for the RACER III is shown in Figure 5-9.

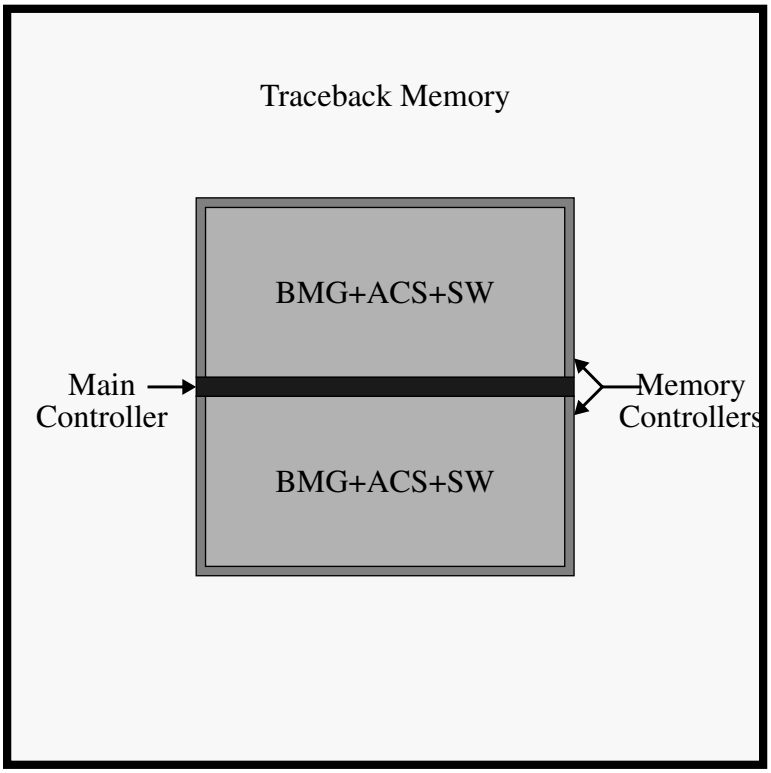



**Figure 5-9:** ***Proposed RACER III Floorplan***

## 5.4 Summary

In this chapter we proposed two new architectures that surpass the Mbit/s barrier. The first design still allowed the reconfigurability of the RACER but relied on faster and denser FPGAs organized in a 16 ring configuration to achieve a data rate of 1 Mbit/s. We showed that depending on the arrangement of the processing stages, a maximum of six signal layers were required to route the state path metrics. This number of routing layers was much smaller than the 28 layers needed to route the BVD. Next, we proposed a fixed Viterbi decoder using a 300 MHz clock that

uses custom VLSI technology to attain a data rate of 6.47 Mbit/s. Though large in size, the RACER III is possible using slightly more advanced technology.

*Chapter* **6**

# Conclusions and Future Work

## 6.1 Summary and Conclusions

We have derived equations to describe the scheduling of any multi-ring cascaded Viterbi decoder. With the aid of these equations, a 41 Kbit/s Reconfigurable Cascaded Multi-Ring Viterbi Decoder (RACER) has been designed, constructed, and functionally verified.

In the course of designing the RACER, faster Mbit/s implementations of the Viterbi decoder were proposed. The first implementation was a 16 ring Viterbi Decoder that took advantage of faster and denser FPGAs. The routing difficulties were discussed when exploiting this kind of parallelism. It was determined that three signal layers would be required for a circular processor layout while six signal layers would be necessary for processors interleaved with each other.

Finally, as technology advances it will be possible to implement Mbit/s long constraint length Viterbi decoders on a single chip.

## 6.2 Future Work

The RACER is not only a constraint length 14 Viterbi Decoder but it can also be adapted for other purposes. Since the RACER is completely reconfigurable, other smaller Viterbi Decoders could be constructed. Moreover, the RACER could be connected to the World Wide Web to permit other researchers access.

Another area of research would be to try and find other architectures that could be mapped into the ring-architecture to be implemented in the RACER. These architectures could include the emulation of communication in a simple hypercube network [Seid86] or mapping a mesh network [Kung88] onto the RACER.

Finally, a search for better codes could be conducted with the help of the RACER.

# References


[Alde83] B. Alder, *Special Purpose Computers,* Academic Press Inc. San Diego, CA, 1988.

[Bert93] P. Bertin, D. Roncin, and J. Vuillemin, "Programmable Active Memories: A Performance Assessment", *Research on Integrated Systems: Proceedings of 1993 Symposium*, MIT Press 1993.

[Bowh95] W. Bowhill, R. Allmon, S. Bell, et. al., "A 300MHz 64b Quad-Issue CMOS RISC Microprocessor", *1995 IEEE International Solid-State Circuits Conference*, February 1995, pp 182-183.

[Coll88] O. Collins, F. Pollara, S. Dolinar, and J. Statman, "Wiring Viterbi Decoders (Splitting deBruijn Graphs)", *TDA Progress Report 42-95*, Jet Propulsion Laboratory, Pasadena, California, Oct-Dec 1988, pp 93-103.

[Coll90] O. Collins, "A VLSI Decomposition of the deBruijn Graph", *TDA Progress Report 42-100*, Jet Propulsion Laboratory, Pasadena, California, February 1990, pp 180-190.

[Coll92] O. Collins, "The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Decoder", *IEEE Transactions on Communications*, December 1992, pp 1810-1819.

[Doli83] S. Dolinar, "A New Code for Galileo", *TDA Progress Report 42-95*, Jet Propulsion Laboratory, Pasadena, California, July-Sept 1988, pp 83-96.

[Feyg90] G. Feygin, *A Multiprocessor Architecture for Viterbi Decoders with Linear Speed-up*. Masters Thesis, University of Toronto, 1990.

[Feyg93] G. Feygin, P. Chow, P. G. Gulak, et. al., "A VLSI Implementation of a Cascade Viterbi Decoder with Traceback", *ISCAS*, 1993.

[FeygM93] G. Feygin and P. G. Gulak, "Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders", *IEEE Transactions of Communications 41*, March 1993, pp 425-429.

[FeygS93] G. Feygin, P. G. Gulak and P. Chow, "A Multiprocessor Architecture for Viterbi Decoders with Linear Speedup", *IEEE Transactions on Signal Processing 41*, Sept 1993, pp 2907-2917.

[Forn73] G. D. Forney Jr., "The Viterbi Algorithm", *Proc. IEEE*, 1973, pp 268-278.

[Fox88] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors Vol I*, Prentice Hall, Englewood Cliffs, N. J. pp 327-348, 1988.

[Gall94] D. Galloway, D. Karchmer, P. Chow, D. Lewis, and J. Rose, "The Transmogrifier: The University of Toronto Field-Programmable System", *Canadian Workshop on Field-Programmable Devices*, 1994.

[Gulak88] G. Gulak and T. Kailath, "Locally Connected VLSI Architectures for the Viterbi Algorithm", *IEEE Journal on Selected Areas in Communications*, 6:527-538, 1988.

[Lin83] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Appllications*. Prentice-Hall, Inc., Engelwood Cliffs, N.J., pp 288-348, 1983.

[Kean94] R. A. Keaney, L. H. C. Lee, and D. J. Skellern, "Implementation of Long Constraint Length Viterbi Decoders using Programmable Active Memories", *11th Australian Microelectronics Conference*, 1994.

[Kung88] S. Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, N. J., 1988.

[Onys89] I. M. Onyszchuk, K. M. Cheung, and O. Collins, "Quantization Loss in Viterbi Decoding Rate 1/n Convolutional Codes", *TDA Progress Report 42-99*, Jet Propulsion Laboratory, Pasadena, California, 1989.

[Onys91] I. M. Onyszchuk, "Testing Interconnected VLSI Circuits in the Big Viterbi Decoder", *TDA Progress Report 42-106*, Jet Propulsion Laboratory, Pasadena, California, August 1991, pp 175-182.

[Stat88] J. Statman, G. Zimmerman, F. Pollara. O. and Collins, "A Long constraint Length VLSI Viterbi Decoder for DSN", *TDA Progress Report 42-95*, Jet Propulsion Laboratory, Pasadena, California, July-Sept 1988, pp 134-142.

[Seid86] S. Seidel and L. Zeigler, "Sorting on Hypercubes", *Proceedings of the Second Conference on Hypercube Multiprocessors*, SIAM, Knoxville, Tennessee, Sept 1986, pp 285-291.

[Vitr67] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, IT-13(4): pp 260-267, 1967.

[Xlxa91] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95124. *The XACT4000-Design Implementation Reference Guide*, pp 7.1-7.23, 1991.

[Xlxb91] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95124. *XACT-Design Interface-Macro Libraries*, pp 3.1-3.105, 1991.

[Xlxc93] XILINX, Inc., 2100 Logic Drive, San Jose, CA 95124. *The Programmable Logic Data Book*, pp 8.110-8.111, 1993.

[Ziem90] R. E. Ziemer and W. H. Tranter, *Principles of Communications Systems, Modulation, and Noise*. Houghton Mifflin Company, pp 1-18 and 676-741, 1990.