# RACER: A Reconfigurable Constraint-Length 14 Viterbi Decoder

David Yeh*, Gennady Feygin+ and Paul Chow

Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road
Toronto, Ontario
Canada M5S 3G4

yeh@eecg.toronto.edu, gfeygin@eecg.toronto.edu and pc@eecg.toronto.edu

## Abstract

*This paper describes the architecture and implementation of a constraint-length 14 Viterbi Decoder that achieves a decoding rate of 41 Kbits/s. The system uses 36 Xilinx XC4010 FPGAs with seven processor cards and a custom backplane to implement a multi-ring general cascade Viterbi decoder architecture. The paper will also show how to achieve decoding rates of 1 Mbit/s using current FPGA technology. Comparisons are made to JPL's Big Viterbi Decoder, which uses custom ASICs.*

## 1. Introduction

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 ratio (SNR) environments. The Viterbi algorithm [Vitr67][Forn73][Lin83] has 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. 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. In this paper, we describe 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.

*David Yeh may now be contacted at: ATI Technologies Inc.
33 Commerce Valley Dr. E.
Thornhill, Ontario
Email: dyeh@atitech.ca

+Gennady Feygin may now be contacted at: Texas Instruments Inc.
MS 8213 PO BOX 65303
Center II, Dallas, TX 75263
Email: a199542@msp.sc.ti.com

## 2. Goals

Since the BVD is the largest Viterbi decoder available, 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 this project were to build a demonstration Viterbi decoder comparable in functionality to the BVD and to achieve the highest throughput possible with the available technology. Moreover, RACER should be reprogrammable as a simulation engine for any other Viterbi decoder up to and including the size of the BVD, and useful for implementing 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 design, simulation, and fabrication of many different ASIC chips time consuming, but they are also very expensive. Moreover, first implementations are not guaranteed to function correctly; thus requiring additional time and money to correct any flaws. SRAM-based FPGAs resolve most of these difficulties. They are relatively inexpensive 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.

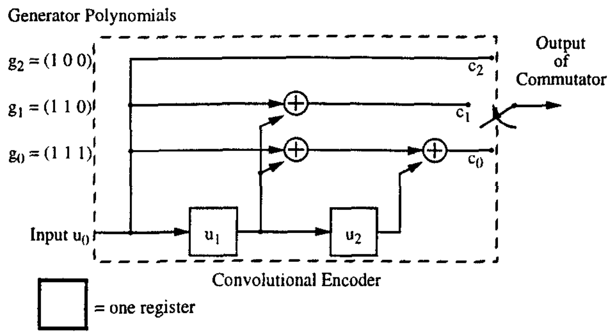


**Fig. 1: A (3,1,2) Convolutional Encoder Example**

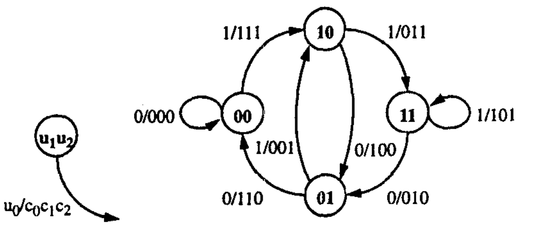


**Fig. 2: Sample State Diagram for Convolutional Encoder**

## 3. Viterbi Background

This section provides some background on the Viterbi algorithm beginning with a description of how data is encoded.

### *3.1. Convolutional Coding*

Data being presented to the Viterbi Decoder are first encoded by convolutional encoders. 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, ν). Figure 1 shows a convolutional encoder for an (n, k, ν) = (3, 1, 2) convolutional code. It accepts one (k = 1) input, or a binary input in this case, and provides three (n = 3) outputs. The three outputs are dependent on the present input, as well as two (ν = 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. In this example, 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.

Operation of the encoder begins with the application of an input $u_0$. 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 $u_0 = 1$ is applied (Figure 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 sequence $u_0 = (1\ 1\ 1\ 0\ 0)$ is applied at the input, the output, (111 001 101 010 100), would be generated.

The above steps can also be illustrated in a state diagram shown in Figure 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 expanded in time to what is known as a trellis diagram (Figure 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 $u_0 = (1\ 1\ 1\ 0\ 0)$.

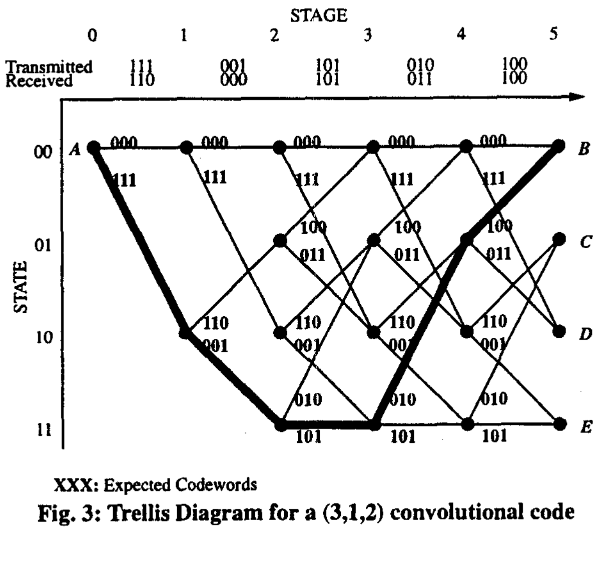
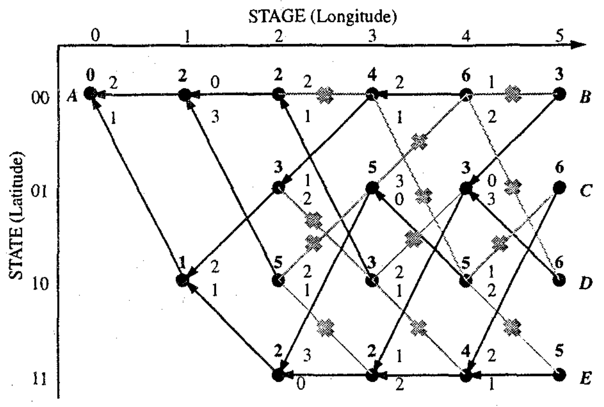


**Fig. 3: Trellis Diagram for a (3,1,2) convolutional code**



*Number* Time Between Intersections (Branch Metric)
*Number* Total Minimum Time at Intersection (Path Metric)
⟵ Selected Path
✖ Discarded Path

**Fig. 4: Viterbi Example**

### *3.2. 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 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 5.

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.

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 at each stage of the trellis 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 the best path in the Survivor Management Unit.

### *3.3. 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 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 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 4 shows the branch metrics for Figure 3 assuming the received codewords are as shown in Figure 3.

### *3.4. 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

```
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 cost top flight = total cost previous top connecting city
                              + cost flying from that city
        total cost bottom flight = total cost previous bottom connecting city
                              + cost flying from that city
        if total cost top flight = total cost bottom flight
          total cost flight = total cost bottom flight
        else
          total cost flight = minimum(total cost top flight, total cost bottom flight)
        end if
      else
        total cost of flight = total cost previous connecting city
                             + cost flying from that city
      end if
      store this total cost 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 gives flight path with the lowest cost
```

**Fig. 5: Algorithm to find the cheapest flight between cities A and B**

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 for the current stage.

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.

### *3.5. 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 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 $3\nu$ to $10\nu$ 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 RE method requires that each state have enough storage to store an entire survivor sequence and the ability to exchange this data between states. This is a common method for shorter constraint lengths. For longer lengths, the TB method is preferred, because it only needs to store one-bit pointers, at the cost of requiring a more centralized memory.

## 4. Previous Work

This section describes the work most relevant to this project. There are two constraint length 14 implementations, the BVD and the DECPeRLe-1 implementation, and a new architecture called the General Cascade Viterbi Decoder.

### *4.1. The Big Viterbi Decoder (BVD)*

The first prototype of the 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

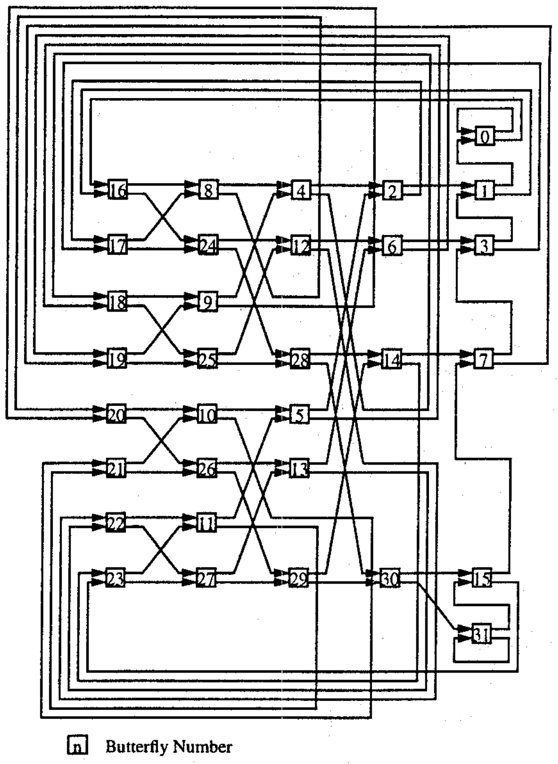


**Fig. 6: Sample Connection of ACS for $\nu = 6$**

built. The second BVD [Onys91][Coll92], comprises 16 identical printed circuit boards with 16 custom VLSI gate-array chips per board, and one board containing the logic for the traceback. Routing in the BVD is extremely difficult as demonstrated by the need to connect the boards with a backplane needing 28 layers! BVD uses 8 Mbit of SRAM for traceback and can attain a decoded data rate of 1 Mbits/s. Connections of the boards and chips are similar to a hierarchical partitioning of a deBruijn graph [Figure 6] for $2^{13}$ butterflies, called a crenellated-FFT [Coll88][Coll90]. By using this form of hierarchical partitioning, longer 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.

### *4.2. DECPeRLe-1 System*

The DECPeRLe-1 [Bert93] is a general-purpose reconfigurable system. It is based on a matrix of 16 Xilinx

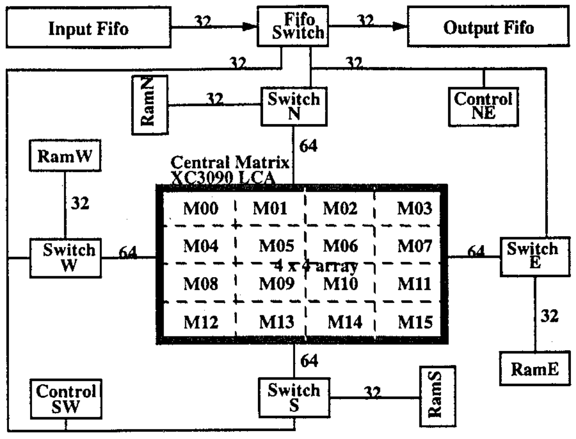


Fig. 7: DECPeRLe-1

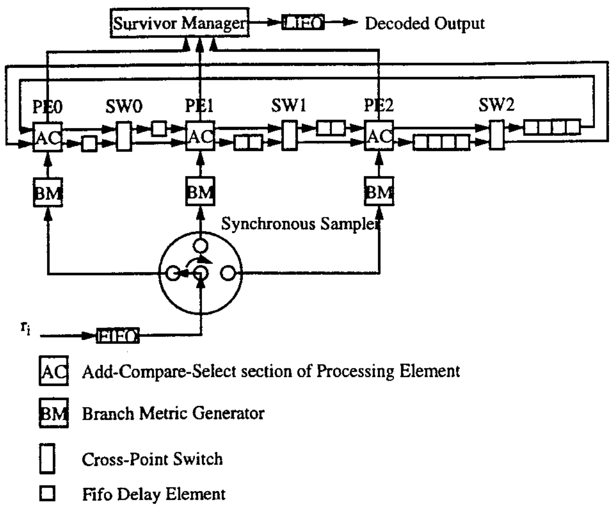


Fig. 8: A Sample GCVD Architecture for ν = 4

XC3090 FPGAs connected to four banks of 32-bit by 256K high speed SRAMS (Figure 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 a decoding rate of 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.

### *4.3. 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 the ACS operations 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 sequentially calculating the path metric information for all $2^{14}$ states. Moreover, time must be spent retrieving and updating path metric information.

An alternative to the above architectures is to use a cascade organization [Gulak88], which is the architecture chosen for the RACER. 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 and BVD, 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.

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 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 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

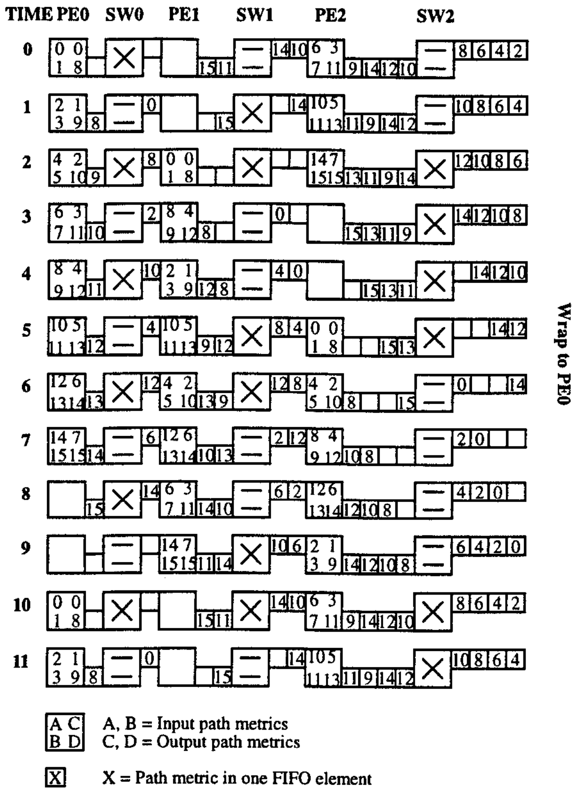

TIME PE0 SW0 PE1 SW1 PE2 SW2
Wrap to PE0
A C / B D A, B = Input path metrics
C, D = Output path metrics
X = Path metric in one FIFO element

**Fig. 9: Timing for a GCVD with a constraint length = 4**

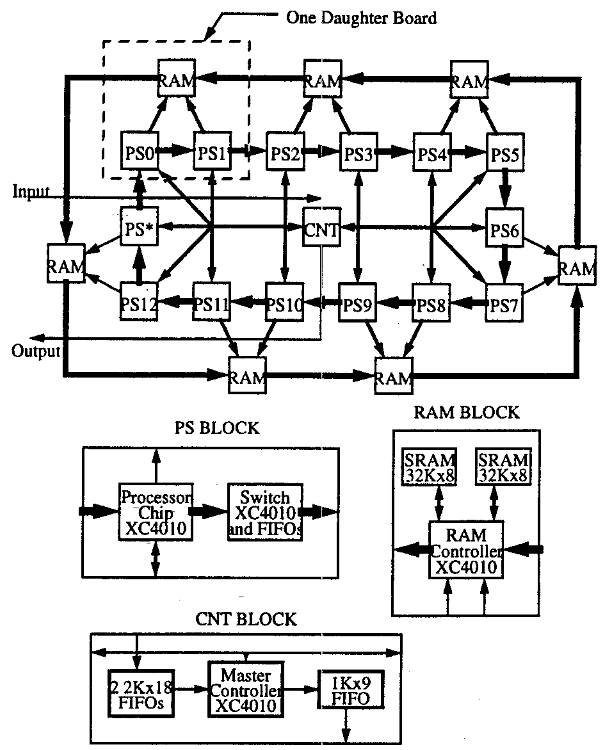


**Fig. 10: Block Diagram of the RACER**

being processed; 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].

## 5. RACER

Implementing the RACER in FPGAs has its drawbacks. The FPGAs currently available are unable to attain 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 previously, path metrics for two states are calculated simultaneously per processing stage, thus requiring a minimum of $2^{v-1}$ clock cycles to complete all $2^v$ 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 $2^{v-1}/2 = 2^{v-2}$. 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 limited hardware available on the XC4010 FPGA allowed only 4 ACS units to be implemented, constraining the RACER to a dual-ring architecture.

### *5.1. Hardware Description*

From a block diagram of the "Flattened" version of the RACER shown in Figure 10, 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 11 shows a block diagram of the backplane. It consists of a Xilinx XC4010 FPGA used as a controller for the

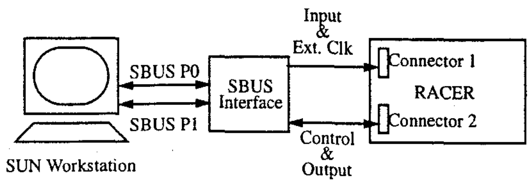


**Fig. 12: SUN Workstation and RACER Configuration**

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 backplane is made up of six layers, four for signals, one for power and one for ground.

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 12.

The physical layout of all the daughter boards 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

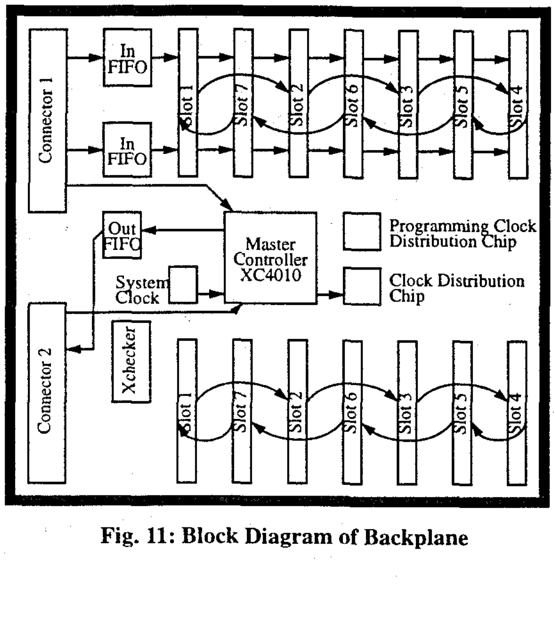


**Fig. 11: Block Diagram of Backplane**

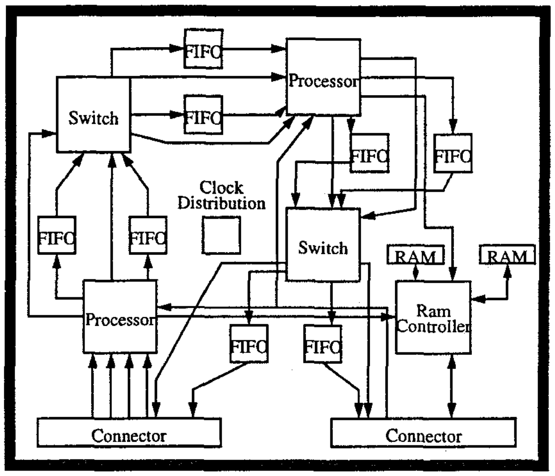


**Fig. 13: Block Diagram of the Daughter Board**

block diagram (Figure 13), 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 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.

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

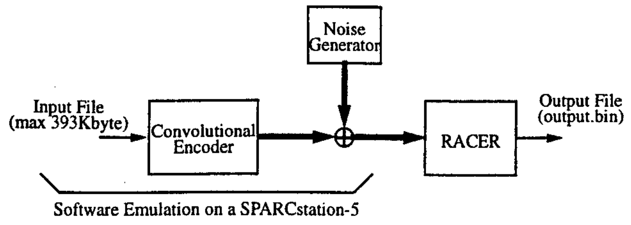


**Fig. 14: Experimental Setup**

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.

### *5.2. 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 14. Through these tests, we have determined that the peak throughput is approximately 41 Kbit/s with a 16 MHz system clock. However, the theoretical data rate should have been 49.6 Kbits/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 quarter 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 1 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 1 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*/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, the next section describes a 1 Mbit/s decoder (RACER II) using FPGAs that would still be cheaper than the BVD.

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

| | Cost (\$ Cdn) | Decode Rate (bit/s) | #Chips for ACS | # of Layers for PCB | Clock (MHz) |
|---|---|---|---|---|---|
| Uni-Processor (SPARCstation-5) | < \$10000 | < 50 | 1 microSPARC II | N/A | 70 |
| BVD Prototype | Multi-million dollar project[a] | 10000 | N/A | 2 Large Racks of PCBs | 20 |
| BVD Prototype II | | 1000000 | 256 custom VLSI chips | 28 | 25 |
| DECPeRLe-1 | ~ \$40000 | 2000 | 16 XC3090 FPGAs | N/A | 25 |
| RACER | ~ \$30000 | > 41000 (Theoretic 49600) | 26 XC4010 FPGAs | 6 | 16 |
| RACER II | ~200000 (estimated) | 1000000 | 104 XC4025 FPGAs | 6 | 48 |

**Table 1: Comparisons of Viterbi decoders based on the BVD specifications**

a. From informal conversations with people in the project.

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.

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 15 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.

## 7. The 1 Mbit/s RACER II

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 decoding rates equivalent to the BVD.

To obtain a throughput of 1 Mbit/s, the RACER II must be at least 20 times faster than the RACER. For a fixed constraint length, the Data Rate may be improved by increasing the clock frequency of the system, and by adding

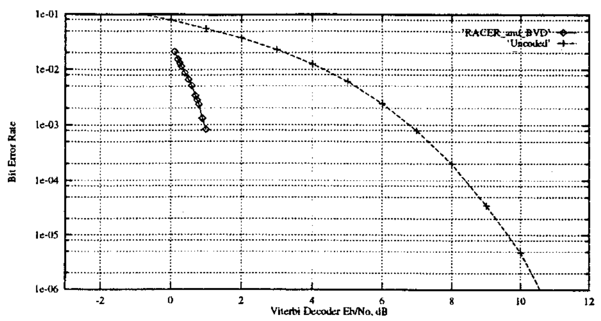


**Fig. 15: Performance Comparison for RACER and BVD**

more rings. 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 332/2 = 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 106496 processing elements which is 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, which is the architecture chosen for the RACER II, is to develop a hybrid of the two extremes.

An important consideration in 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, a 16 ring cascaded Viterbi decoder is sufficient for a 1 Mbit/s decoding rate. With this architecture, the minimum system speed is 46.4 MHz. With the introduction of more dense FPGA technology such as the XC4025E FPGA, this goal is certainly achievable.

Assuming the same general architecture as the RACER, a rough estimate of the total cost of the RACER II can be determined. If the cost of the new two-sided boards, labour and other components are comparable to the original total cost of the RACER, then the only cost to add is the price of the XC4025E, which is estimated to be about $1500 each at today's prices. 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 $200,000, which is 10 times cheaper than the BVD at the same decoding rate and without the need for custom ASICs. This system is also much less complex and could be implemented by a single Master's student in about a year.

## 8. Conclusions

A 41 Kbit/s Reconfigurable Cascaded Multi-Ring Viterbi Decoder has been designed, constructed and functionally tested. The RACER is not a fixed length Viterbi Decoder but it is reconfigurable for any other smaller Viterbi Decoder. The system can also be used for other applications that have this ring-like architecture.

A 1 Mbit/s version of the RACER has been proposed that has significantly less routing than the BVD. Moreover, with the aid of the fastest, and most current FPGAs, an equivalent system can be produced with a shorter development time and lower cost than the BVD which uses four year old ASIC technology.

Although FPGAs will not directly compete in speed and area with custom ASICs implemented with the latest technologies, this project has shown that by using architectures that are suited to the FPGA technology, it is possible to build cost-effective systems competitive with systems using ASIC technologies. With the current trends in access to fabrication, it will also become difficult to access the best technologies unless there is sufficient volume, in which case FPGAs would be competitive alternatives for low-volume applications compared to ASICs implemented in slightly older technologies.

## 9. Acknowledgments

We would like to acknowledge Xilinx and Motorola for donating most of the components, in particular, Xilinx for their ongoing support of FPGA work at the University of Toronto. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

## 10. 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.