A VLSI Implementation of a Cascade Viterbi Decoder with Traceback

Gennady Feygin Paul Chow P. Glenn Gulak John Chappel Grant Goodes
Oswin Hall Ahmad Sayes Satwant Singh Michael B. Smith Steve Wilton

Department of Electrical and Computer Engineering
University of Toronto

Abstract — A novel VLSI implementation of the Viterbi algorithm based on a cascade architecture is presented. Survivor sequence memory management is implemented using a new single read pointer traceback technique. The overall design for a 16-state, rate 1/2 decoder requires about 26000 transistors and a core area of 8.5 mm^2 in a 1.2 μm two-level metal CMOS technology.

1 Introduction

As the use of digital communications grows, advanced coding techniques and increased throughput are needed. Higher clock rates are required or the parallelism inherent in the decoding algorithm must be exploited in multiprocessor implementations.

Recently there has been increased interest in multiprocessor implementations of decoders for the class of error-correction codes known as convolutional codes. In particular, decoders that implement the Viterbi algorithm [1] are of interest.

A number of high-performance multiprocessor architectures have been proposed for Viterbi decoders, including one- and two-dimensional systolic meshes and the perfect shuffle layout [2, 3]. All of these architectures provide a throughput increase that is sub-linear (i.e., to increase the throughput by a constant factor more than a constant growth in area is required [3]). An architecture that is closely related to the perfect shuffle, known as the crenellated-FFT [4], is currently being implemented at the Jet Propulsion Laboratory in the receiver for the Galileo space probe.

The Viterbi decoder described in this paper demonstrates the viability of a "cascade " family of Viterbi decoder architectures [5] that employ small-to-medium scale parallelism to achieve a linear increase in throughput with only a linear increase in silicon area required, when compared with a uniprocessor implementation. For a 2^ν-state decoder, the cascade architecture implemented here uses $\nu - 1$ processors, as compared with a fully parallel architecture that uses 2^ν processors. The cascade architecture is targeted towards applications where a fully parallel decoder is not practical (large constraint lengths; $\nu \geq 8$). For these applications the cascade Viterbi decoder is considerably more area-efficient than other multiprocessor Viterbi decoder architectures [5, 3] that typically exhibit quadratic increase in silicon area with linear increase in throughput.

Viterbi decoders have two major components: an *add-compare-select* (ACS) data-path, and a survivor sequence memory management unit. In the ACS data-path, path metrics are updated by adding branch metrics associated with each possible state transition. The smaller path metric is the new path metric for the state and the resulting decision is stored in the survivor sequence memory management unit where the most likely path is determined.

The Viterbi decoder described here uses a ring-connected ACS data-path structure with high regularity. Deep pipelining inside each *add-compare-select* (ACS) unit combined with local unidirectional wiring in the ACS data-path results in a better throughput per unit area [5] compared to other known implementations[1]. For the survivor sequence memory management, a novel traceback technique with multiple memory modules and a single read pointer [6, 10] is utilized. Figure 1 depicts the general structure of the decoder.

2 ACS Data-path

When describing a convolutional code with a trellis diagram the nodes correspond to the Add-Compare-Select (ACS) computations and the edges indicate dependencies between the ACS computations of the successive stages of the trellis. Given a number of individual ACS units we can derive a large number of possible assignments of the nodes of the trellis diagram to the physical ACS units. In this paper we concentrate on an assignment where all computations in a given stage of the trellis are performed on a single ACS unit. This particular assignment characterizes the cascade family of Viterbi decoder architectures. There is also a schedule of state updates that can be de-

[1]The VLSI decoder being presented was intended as a small-scale demonstration of the superior features of the cascade architecture (modularity, local communications); however pipelining of the ACS units is not possible for $\nu = 3$ [5].

0-7803-1254-6/93$03.00 © 1993 IEEE

scribed as follows: The *butterfly i* is a set of two ACS operations that "consume" the path metrics of the states $2i$ and $2i+1$ from the previous stage and "produce" the path metrics of the states i, $2^{\nu-1}+i$. The order of butterfly computations is $0,1,2,3,4,5,6,7$ in the first ACS, $0,4,1,5,2,6,3,7$ in the second ACS, and $0,2,4,6,1,3,5,7$ in the third ACS unit. The butterfly computations must be staggered in time, with the butterfly operations in the second ACS starting two clock cycles later than in the first, and the butterfly operations in the third ACS starting three clock cycles later than in the second. Each processor remains idle for two clock cycles following each set of eight butterfly operations.

The cascade Viterbi decoder with ν processors connected into a ring is described in [3]. More recent work has studied general properties of all possible cascade Viterbi decoders with an arbitrary number of processors [5]. It can be shown that the architecture that employs exactly $\nu-1$ processors has the highest possible throughput from among all possible choices of the number of processors. Furthermore, the cascade architecture with $\nu-1$ processors requires the least complex cross-point switches to achieve correct scheduling of the computations.

The main features of the cascade Viterbi decoder architecture with $\nu-1$ processors are:

- Highly regular and modular structure.
- Processors are connected in a ring with local unidirectional communication.
- High processor utilization, which improves rapidly for designs with larger constraint lengths.
- Pipelining inside each processor is available, with the pipelining depth improving rapidly for designs with larger constraint lengths. Availability of pipelining inside each processor allows low complexity ripple-carry adders to be used, improving regularity and further reducing the area required.

As path metrics advance around the ring of processors, path metric values accumulate and will eventually cause an overflow. We use the scheme for controlling overflow described in [7], which is widely used in industry and well suited for our design.

The ACS operation is made faster by allowing the Compare operation to proceed in parallel with the Add operation as soon as the LSB of the Add operation is available. The Compare operation remains one bit behind the Add operation, so that at the moment when the Add operation is completed, the Compare operation has only a one-bit (MSB) compare to perform. The hardware required to implement the Compare consists of the carry chain of an adder, with a sum cell required only for the MSB.

Though branch metrics are computed as the Euclidian distance between the possible transmitted codeword and the received codeword, a simplification suggested in [8] allows us to avoid the necessity of the multiply operation.

3 Survivor Management

In a Viterbi decoder, there are two known memory organization techniques for the storage of survivor sequences from which the decoded information sequence is retrieved, namely *register exchange* and *traceback* [9]. Our implementation uses traceback.

The traceback method stores path information in the form of an array of recursive pointers. It is advantageous to think of traceback memory as organized in a two-dimensional structure, with rows and columns. The number of columns is equal to the number of states $N=2^{\nu}$. Each row stores the results of N comparisons corresponding to one symbol interval or one stage in a trellis diagram. Since the stream of symbols is, in general, semi-infinite, storage locations are periodically re-used. Enough rows must be provided to guarantee minimum a traceback depth of $T \geq 10\nu$. There are three types of operations performed in parallel inside a traceback decoder: writing new decision data into memory; and two read operations — traceback read and decode read. During either read operation previously stored decisions are used as pointers to re-construct the state trajectory in the trellis. Decision bits that have been read during the traceback operation are not output, but rather used to guarantee high probability of "path merging". The final state of the traceback trajectory becomes the the starting point of the subsequent decode read operation. Decision bits read during the decode read operation are sent to the LIFO register for order reversal and finally output as the decoded bits.

For every set of column write operations (N bits wide), *an average* of one decode read must be performed. The overhead of the T-column traceback read can be spread over more than one column decode read operations. This includes both decode read operations and traceback read operations. In our design we have implemented a one-pointer variant [10].

4 Implementation

To determine the viability of the cascade architecture, a 16-state, rate 1/2 decoder was implemented using a 1.2 μm two-level metal CMOS technology. Figure 2 is a die photo of the chip. Fabricated chips have been received and shown to be functional up to a clock frequency of 32 MHz at room temperature. The maximum frequency attained was limited by the tester. Conservative design, using fully static circuits and a focus on testability, was used for this chip since it was implemented during a one-term course

project. A more aggressive design will be capable of much higher clock rates.

The floor-plan of the chip closely approximates the architecture shown in Figure 1 except that the branch metric generators (BMG) are aligned in the data-path with the ACS units. The top half is the traceback unit (TB) and LIFO, and the bottom half is the ACS data-path. The ACS data-path handles data quantized to four bits and uses eight-bit path metrics. There are three processors and each processor implements two ACS operations as shown in Figure 3. For a clock rate of 32 MHz, this results in a data rate of about 10 Mbits/s.

The organization of the traceback unit is shown in Figure 4. In this implementation three 28×16 RAM modules are used to store ACS decisions. They are first clocked into shift registers and then written into the RAMs when one trellis stage has been completed, $2^\nu = 16$ bits at a time. During each traceback, 14 trellis stages are read from each of the three modules. This provides a traceback depth of $42 \approx 10\nu$ for a 16-state trellis. Though dual-port memory modules were used in this design, single-port memory modules would scale better for large constraint length decoders [10].

Next to the RAMs are the read and write pointers (implemented as one-bit wide circular shift registers) that are used to select word lines in the RAMs (each word line corresponds to one stage of the trellis, or 16 bits). Read and write pointers move through the memory in opposite directions, with wraparound modulo-28. In addition to the read pointer and the write pointer there is a 4-bit state pointer and a last-in first-out (LIFO) register. The state pointer is updated as follows: when a 16-bit word-line is read, four bits of state pointer are used to control the 16-to-1 multiplexor. One of two possible predecessor states is now selected by a one-bit shift to the left (with MSB discarded) and adding the bit from the output of the 16-to-1 multiplexor to the LSB position. This operation is repeated at the other memory banks until $\nu - 1$ shifts and bit updates have been performed and a resulting ν-bit state latched. The latched ν bits are made available at the input to the LIFO; they are either ignored (during the traceback) or loaded into the LIFO and re-ordered (during the decode) to generate the decoded output. It may appear that the traceback and decode operations are the speed bottleneck, since three memory reads occur for every memory write. This however is not a problem for traceback decoders with longer constraint lengths, since the number of state update reads grows proportional to $\nu - 1$, while the number of state decision writes grows proportional to 2^ν. As the word-length grows exponentially, a point is rapidly reached where the designer is forced to limit word-length growth and perform state decision writing in multiple cycles; consequently, writing the decisions, not reading ultimately limits the speed.

For testing, all of the ACS data-path registers are linked into a scan chain and provision has also been made to permit reading and writing of the traceback memory. There are a total of 54 pads with 22 pads devoted to testing making the chip pad limited.

5 Summary

The design described in this paper is highly modular, with only regular, local wiring required. As discussed in Section 2, available pipelining and processor utilization both improve rapidly for larger constraint lengths. At the same time, all high-speed on-chip data transport (processor-to-processor, processor-to-decision memory, and branch metric generator-to-processor) is local and uniform, with simple routing requirements. Our design can be expected to scale nicely to larger constraint lengths.

The implementation of a cascade Viterbi decoder architecture, which achieves a linear increase in throughput with a linear increase in silicon area compared to a uniprocessor, has demonstrated that this architecture is well-suited to VLSI implementation. This architecture is particularly well suited for applications where full-scale parallelism (2^ν processors) is either not required due to a low data rate, as in low speed modems operating on high SNR channels, or is impractical or prohibitively expensive due to the fact that the number of processors (2^ν) is very large, as in low SNR channels where the constraint length is large. A novel traceback architecture using a single-read pointer was implemented as a part of this chip.

References

[1] G. D. Forney Jr. The Viterbi Algorithm. *Proc. IEEE*, 61:268–278, March 1973.

[2] C. Y. Chang and K. Yao. Viterbi Decoding by Systolic Array. In *Proceedings of the Twenty-third Annual Allerton Conference on Communications, Control and Computing*, pages 430–439, Monticello, Illinois, October 1985. Allerton House.

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

[4] O. Collins, F. Pollara, S. Dolinar, and J. Statman. Wiring Viterbi Decoders (Splitting de Brujn Graphs). TDA Progress Report 42-96, Jet Propulsion Laboratory, Pasadena, California, October-December 1988.

[5] G. Feygin. A Multiprocessor Architecture for Viterbi Decoders with Linear Speed-up. M. A. Sc. thesis, University of Toronto, Toronto, Canada, 1990.

[6] G. Feygin, P. G. Gulak, and F. Pollara. Survivor Sequence Memory Management in Viterbi Decoders. In *Third IBM Workshop on ECC*, San Jose, California, September 1989.

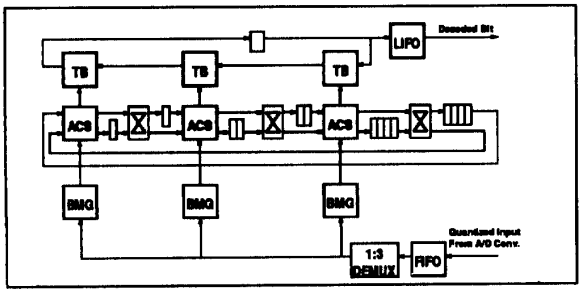

Figure 1: Architecture of the cascade Viterbi decoder.

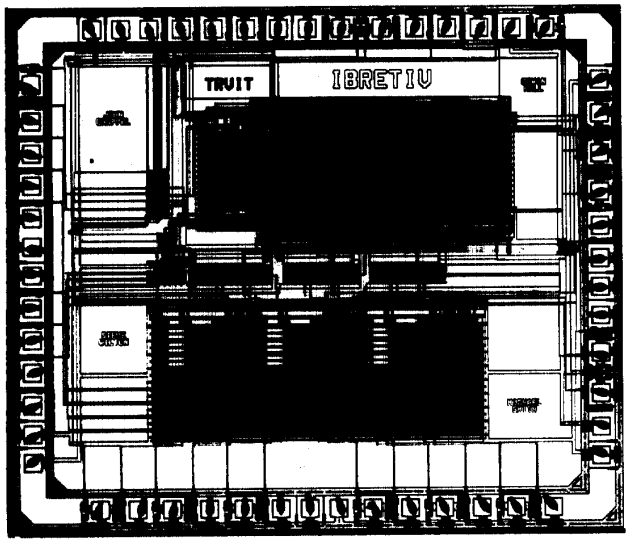

Figure 2: Die photo of the 3-processor Viterbi decoder.

[7] A. P. Hekstra. An Alternative to Metric Rescaling in Viterbi Decoders. *IEEE Transactions on Communications*, 37:1220–1222, November 1989.

[8] K. S. Gilhousen et al. Coding systems study for high data rate telemetry links. Technical report, NASA, January 1971. Prepared by Linkabit Corp. under contract NAS2-6024.

[9] C. M. Rader. Memory Management in a Viterbi Algorithm. *IEEE Transactions on Communications*, 29:1399–1401, September 1981.

[10] G. Feygin and P.G. Gulak. Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders. *Accepted for publication in IEEE Transactions on Communications*, 1991.

Acknowledgments

The authors gratefully acknowledge financial support provided by the NSERC of Canada and by the ITRC of Ontario. Fabrication was provided by the Canadian Microelectronics Corporation. Thanks go to Lysander Lim for testing the chip.

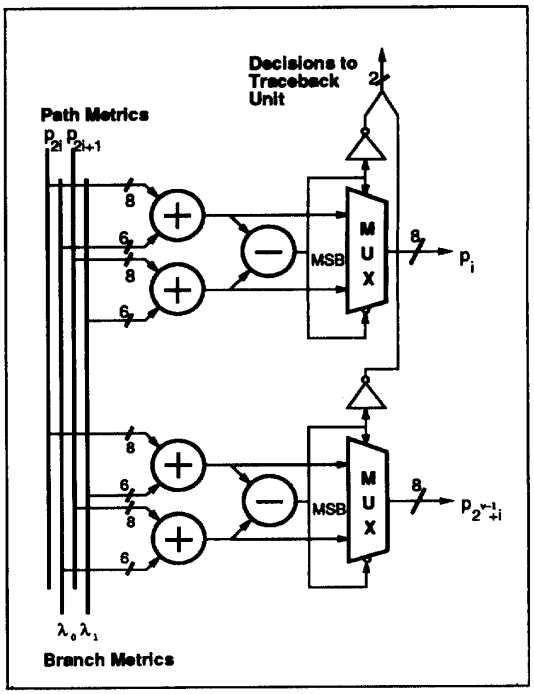

Figure 3: Architecture of the ACS unit.

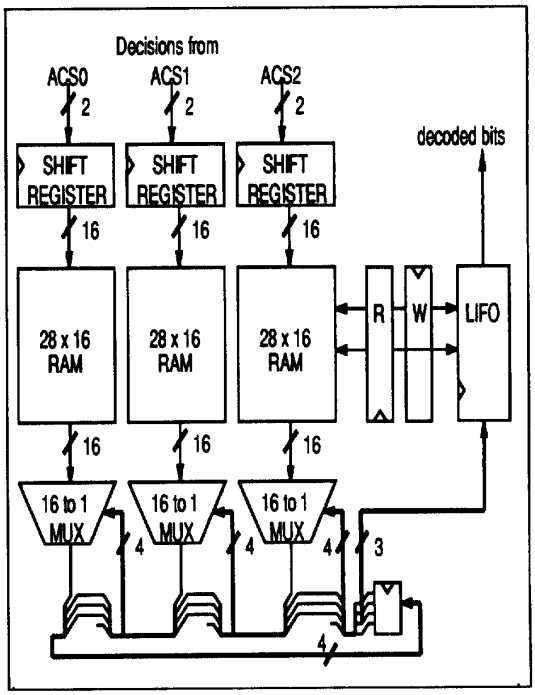

Figure 4: Architecture of the traceback unit.