A Reconfigurable Viterbi Decoder Architecture

Kanu Chadha and Joseph R. Cavallaro
ELectrical and Computer Engineering Department
Rice University
Houston, TX, 77005
{kanu,cavallar}@rice.edu

Abstract

We present the design and implementation of a novel reconfigurable Viterbi decoder which provides dynamic adaptation to different constraint length and code rate convolutional codes. A decoder that supports constraint lengths from 3-7, and code rates 1/2-1/3 has been synthesized on an FPGA. With a throughput of 20 Mbps, the proposed decoder is suitable for use in receiver architectures of the 802.11a wireless local area network and 3G cellular code division multiple access environments. Results show that the area overhead associated with such a reconfigurable implementation as compared to a fixed constraint length 7 implementation is just 2.9%.

1 Introduction

Forward error correction schemes, typically convolutional coding, are an essential component of today's wireless communication systems. Present wireless standards such as the third generation (3G) systems and the 802.11a WLAN utilize some formulation of convolutional coding, which is usually decoded via Viterbi decoders. Each standard specifies a different configuration for the convolutional code which imparts different requirements on the decoder. For example, the 802.11a standard specifies a constraint length 7, rate 1/2 code whereas the convolutional code for the 3G standard has a specification of constraint length 9 and rate 1/3. On the other hand, there is a growing interest in building devices or systems that can operate on multiple standards and gain benefit through reuse of hardware. This becomes especially important in the context of future multi-tier networks which aim to support different kinds of networks on a single device and provide the capability of seamlessly switching across networks. This motivates the need for a Viterbi decoder which has the capability to dynamically switch standards.

Although Viterbi decoding can be done in a flexible manner on the DSP [5], realizing flexibility in hardware is a challenging problem. A hardware solution[6, 8] is, however, significantly more cost effective and capable of producing higher decoding throughput than a software solution. This paper presents a flexible Viterbi decoder hardware solution that can operate for multiple constraint lengths and code rates. This flexibility has been achieved by adopting a fully parallel approach for the highest constraint length and code rate. The second section reviews the basic components of a Viterbi decoder. The third section details the proposed solution for a reconfigurable architecture. A prototype implementation for constraint lengths of 3 through 7 and code rates 1/2 to 1/3 is presented in the fourth section and we conclude our discussion in the last section.

2 Viterbi Decoding Review

Viterbi decoding is a technique for performing maximum likelihood sequence detection on data that has been convolutionally coded. The decoding problem is to determine the path with the minimum path metric through the trellis, with path metric being defined as the sum of the branch metrics along the path. This is done in a stepwise manner by processing a set of state metrics forward in time, stage by stage over the trellis.

The complexity of the Viterbi algorithm lies in the computation of 2^{K-1} path metrics for a constraint length K decoder at each time stage. For the rate $1/n$ codes we are considering, there are just two predecessor states or branches for each state. Thus, state metric computation involves calculation of two branch metrics per state and then a selection of that branch which gives a smaller value of the new state metric. The former operation is done in the Branch Metric Unit(BMU) which takes in the received n-bit blocks of data and generates branch metrics by computing the distance between the received data and the actual codewords. The latter selection operation is performed by the Add Compare Select (ACS) unit. The

0-7803-7147-X/01/$10.00©2001 IEEE

ACS unit takes in two state metrics and two branch metrics as input to yield an updated path metric. As the above process is performed, the selected or surviving branches for each state are recorded by storing one survivor bit per state at each trellis stage. The Survivor Management Unit (SMU) is responsible for tracing back through the trellis using the survivor bits to produce the input data bits.

Thus, the essence of the Viterbi algorithm lies in the relatively simple operations of add, compare, select, and traceback which need to be applied to a large number of states. This motivates a fully parallel[2, 4] processing approach which was mentioned earlier. In this approach, each state is assigned one processor leading to the highest possible decoding throughput but a large area utilization and a complicated interconnection network. At the other extreme, a uniprocessor implementation though poor in speed, requires a very small area. There exist intermediate alternatives[1] such as the general cascade approach, which uses K processing elements, and provides decoding speed better than the uniprocessor but slower than the fully parallel implementation.

3 The Proposed Architecture

Considering the data rate requirements for the next generation 802.11a (6-54 Mbps) and 3G (2.5 Mbps), we selected a fully parallel implementation approach for all constraint lengths. This means that there is hardware for the highest constraint length and code rate while the smaller constraint length, lower code rate decoders use only part of the total resources available. Also, the parallel approach obviates the need for any state metric memory and corresponding address generation circuitry making the design simpler. However, we do have to incur the complexity of the state interconnection network.

A major contribution of this paper is the identification of a component, hereafter referred to as the Viterbi core, which forms the basic building block of the Viterbi decoder, given any constraint length and code rate. The Viterbi core, shown in Figure 1, is designed to operate on a butterfly, a basic element of the trellis which is shown in Figure 2. The Viterbi core is composed of one BMU, two registers for storage of path metrics, an ACS unit which updates two states of a butterfly, and optional multiplexors at the input of the path metric registers. There are 2^{K-2} butterflies in the trellis of a constraint length K decoder. The Viterbi core can be replicated as many times as dictated by the constraint length of the code to yield different configurations of the decoder. For example, for a $K = 5$ decoder, with 16 states and 8 butterflies, the Viterbi core needs to be replicated 8 times. Note that the *selectK* and *selectR* lines are inputs to the Viterbi core for determining the mode of operation, i.e. the constraint length and code rate of the decoder. The detailed operation of the Viterbi core is described in the following sections.

3.1 Architecture of BMU core

The BMU core is responsible for calculating 2^K branch metrics at each stage of the trellis. Each branch metric is computed as the Hamming distance between the received n-bit block and the actual codeword. Some inherent symmetry in the trellis is exploited to simplify these computations. We make the following observations from the trellis structure.

- There is only one codeword per butterfly computation.
- There is always a branch from state $2i$ to state i irrespective of the constraint length of the code.

Thus, we can dedicate one BMU core unit per butterfly to compute the unique branch metric per butterfly. Further, since BMU core i needs to compute a branch metric for the transition from state $2i$ to i, each BMU core needs to be initialized to a different state. The BMU core shall be internally composed of a single shift register encoder which computes the relevant codeword and some additional circuitry to calculate the distance between the received word and the codeword. In order to support multiple code rates, the BMU core unit shall have hardware for the highest code rate. Note that the effect of supporting multiple code rates is felt only in the BMU while changing K modifies the trellis diagram and thus, effects the whole architecture.

3.2 Architecture of ACS core

This is the computationally most demanding part of the Viterbi decoder. The ACS core has inputs as two branch metrics, and two path metrics and outputs as two updated path metrics and two decision bits. The results of the path metric updates are stored back to registers while the decision bits are written to survivor memory. The interconnection network between the ACS units and the path metric storage registers is very complex and highly dependent upon the constraint length. In general, for a typical butterfly calculation, as seen previously in Figure 2, the input states are $2i$ and $2i + 1$, and the output states are i and $i + 2^{K-2}$. This shows the dependence of the metric updates on the constraint length. So, for the fully parallel architecture we are considering, we needed to find a flexible way to route the outputs of the ACS

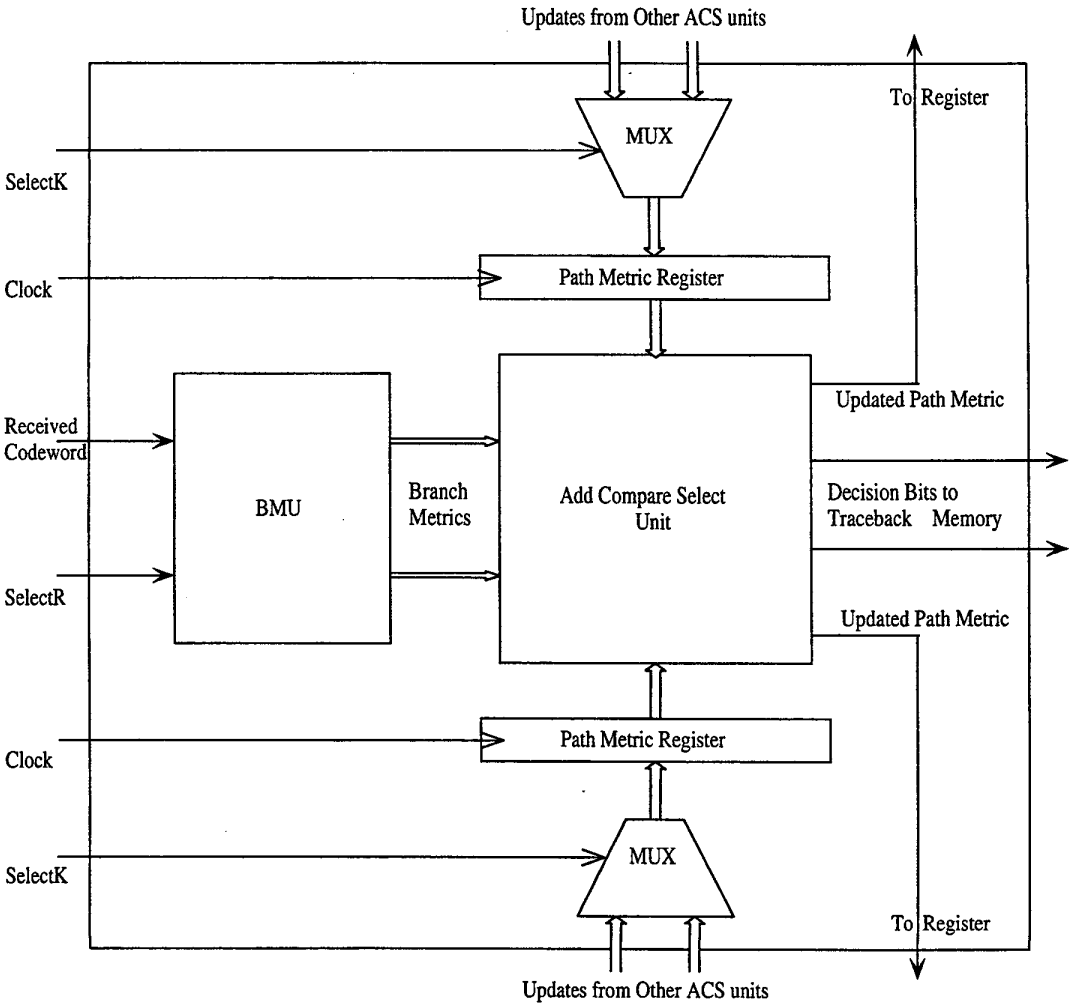

Figure 1: The Viterbi Core

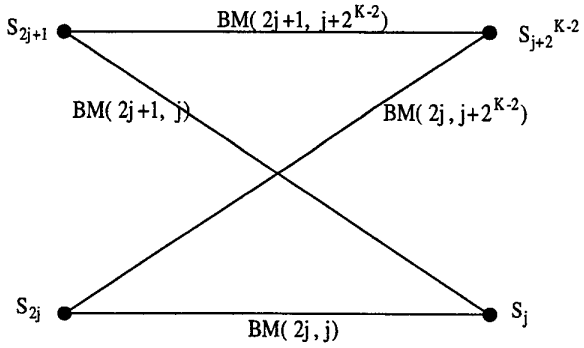

Figure 2: A Typical Butterfly

units to the appropriate path metric registers for all constraint lengths.

Let us suppose that we want to support a range of constraint lengths from $[K_1, K_2]$, $K_1 < K_2$. So, we have 2^{K_2-2} ACS units and 2^{K_2-1} registers for path metric storage. Let us index the outputs of the ACS units as [ACS(0), ACS(1), ..., ACS(2^{K_2-1})], ACS($2i$) and ACS($2i+1$) being the outputs of ACS unit i; and the inputs to the path metric registers as [PM(0), PM(1), ..., PM(2^{K_2-1})]. From the trellis diagrams for different constraint lengths, we make the following important observation: PM(i) can be fed from either ACS($2(i-2^{K-2})+1$) or ACS($2i$) depending upon the value of K. Equations 1 to 3 describe the interconnections between the outputs of the ACS units and the inputs to the path metric registers as a function of K.

for $0 \leq i < 2^{K_1-2}$

$$PM(i) = ACS(2i) \ \ \forall K \tag{1}$$

$$PM(i) = \begin{cases} ACS(2(i-2^{K-2})+1); \\ \quad \text{choosing K } s.t. \text{ (i - } 2^{K-2} \geq 0) \text{ is} \\ \quad \text{smallest } \forall K \in [K_1, K_2) \\ ACS(2i) \qquad\qquad \text{; otherwise} \end{cases} \tag{2}$$

for $(2^{K_2-2} \leq i \leq 2^{K_2-1})$

$$PM(i) = ACS(2(i-2^{K_2-2})+1) \text{ for K} = \text{K}_2 \tag{3}$$

In this way, by using a 2-input multiplexor to feed the path metric registers, and controlling the output of the multiplexor by the *selectK* input of the decoder, we can route the ACS outputs to the correct path metric registers. The multiplexors are the additional overhead which is needed to make the interconnection network reconfigurable. Apart from the path metrics, the ACS units also produce two decision bits for the the two states of a butterfly. A decision bit is 1 if the surviving branch comes from a state with higher index and zero otherwise. These decisions are used by the Survivor Management Unit to reconstruct the sequence of states the decoder goes through. Decision bits for all the states are produced simultaneously and must be written to memory in parallel.

3.3 Survivor Management

The survivor management unit is responsible for reconstructing the sequence of input bits from the decision bits generated by the ACS units. It consists of a survivor memory which holds the decision bits and a controller for the memory. The survivor memory size is chosen according to the largest constraint length decoder. For a decoder which needs to support a maximum constraint length of K, the survivor memory must have a width of 2^{K-1} bits.

Each of the 2^{K-1} bits is written to by the decision bits produced by the ACS units at each stage of the trellis. Each bit position represents a state in the trellis diagram, and each location in memory represents a time step in the decoding process. When writing the decision bits into memory, the same problem occurs as in writing the updated path metrics to the registers. This is because the outputs of the ACS units again need to be written to different bit positions in memory depending upon the constraint length. This problem is solved in a similar manner by using 2-input multiplexors at the data input to the survivor memory.

There are two well known approaches to survivor management, namely register exchange and traceback[7]. We chose the traceback scheme over the register exchange method because the traceback is more suitable for reconfiguration purposes. This is because of its more "programmable" approach as compared to the "hardwired" approach of the register exchange method. The controller of the survivor memory contains selection hardware to switch between constraint lengths as dictated by the *selectK* input to the SMU.

4 Prototype Implementation and Results

A prototype version of the architecture described above was implemented using Field Programmable Gate Array (FPGA) technology (our system used the Xilinx XCV800 FPGA). The architecture was described at a behavioral level using VHDL. The implemented Viterbi decoder can operate for constraint lengths of 3 through 7 and code rates 1/2 and 1/3. Our Viterbi decoder implementation is currently packet based, i.e. it waits for the reception of the entire incoming message before it starts decoding.

4.1 Implementation

Shown in Figure 3 is the block diagram of the reconfigurable Viterbi decoder. The main components of the decoder are the controller unit, the Viterbi core, the interconnection network and the traceback memory. Also part of the design are configuration registers which hold parameters such as the various generator polynomials, and the different values of the constraint lengths and code rates. The reset input is important because it is used in conjunction with the *selectK* and *selectR* input to reconfigure the decoder configuration. The decoder requires just one clock cycle for reconfiguration purposes. The controller takes in the control inputs and generates the appropriate signals for the ACS, BMU and the traceback memory. It configures the BMU through the *selectR* input, manages the ACS

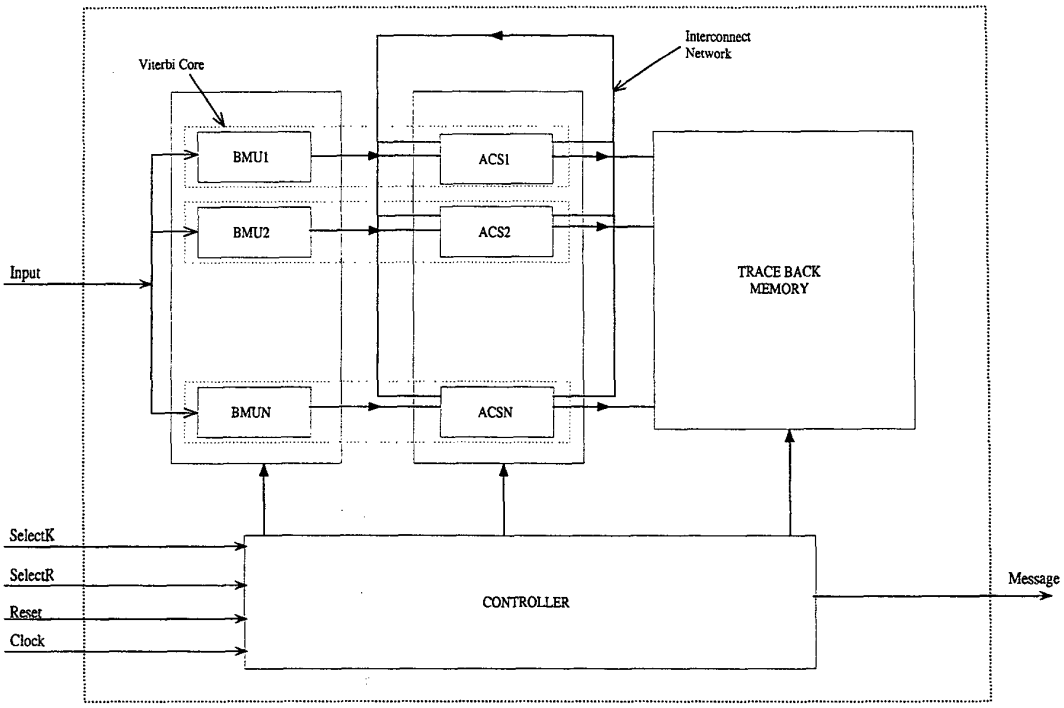

Figure 3: The Viterbi Decoder Implementation

interconnections through the *selectK* input and controls the reading/writing of traceback memory. The traceback memory was implemented using the built-in RAM blocks of the Xilinx FPGA. The XCV800 FPGA has 28 4096-bit Synchronous RAMs each of which can be configured to have port widths from 1, 2, 4, 8, up to 16 bits. We concatenated 4 such RAM blocks to create a 64-bit wide traceback memory.

4.2 Results

To compare the FPGA resources consumed by the reconfigurable decoder implementation with those utilized by a fully parallel static implementation, we have summarized the equivalent gate count for the reconfigurable case and for the corresponding fixed one in Table 1. We can see that the gate count for the reconfigurable decoder is slightly higher because of the overhead associated with the multiplexors and the control logic that manages the reconfiguration. However, the overhead is just 2.9% even when we compare a rate 1/2-1/3, $K = 3-7$ decoder with a corresponding fixed implementation.

There is little degradation in performance in terms of the decoding throughput since the implementation is fully parallel for all constraint lengths. The throughput of 20 Mbps achieved by such an FPGA implementation is much larger than that attainable by the most powerful current DSPs. The most recent TI TMS320C6416 DSP with a Viterbi coprocessor can provide a maximum of about 4 Mbps decoding throughput. Moreover, the throughput in our case can be increased three folds to about 80 Mbps if we change the decoder implementation to a streaming one instead of packet based, at the cost of additional traceback memory.

Table 1: *Reconfigurable Vs Fixed: Throughput/Area*

K	Area(gates)	Overhead(%)	Mbps
5	23,055	-	20.1
$3-5$	23,481	1.8	19.9
7	86,845	-	20
$3-7$	89,407	2.9	19.7

It should be remembered that the implementation described above is a prototype version and as such the performance and size is determined and limited by the present FPGA technology.

5 Conclusions

We have proposed a new Viterbi decoder architecture which allows for run-time adaptation to different code configurations, i.e. set of constraint lengths and code rates. The new architecture is a fully parallel structure for all configurations and thus suitable for very high data rate decoding. The architecture was implemented on a Xilinx FPGA. The area overhead associated with a reconfigurable decoder implementation which can support constraint lengths 3 to 7, when compared to a static constraint length 7 decoder is just around 2.9%. Thus, our design does not consume much more resources than an equivalent non-reconfigurable decoder but yields the facility of run-time reconfiguration. Moreover, the throughput achieved with the FPGA implementation is much higher than that possible by using current DSPs. We believe that such an architecture has considerable application in future networks where different decoding standards shall have to be supported on a single device.

6 Acknowledgements

This work was supported in part by Nokia Corporation, Texas Instruments Inc., the Texas Advanced Technology Program under grant 1999-003604-080, and by NSF under grant ANI-9979465.

References

[1] P.G Gulak and T. Kulaith. Locally Connected VLSI Architectures for the Viterbi Algorithm. In *IEEE Journal on Selected Areas of Communications*, 6(3):527-537, Apr 1988.

[2] D. Yeh, G. Feygi, and P. Chow. RACER: A Reconfigurable Constraint Length 14 VIterbi Decoder. In *The Fourth IEEE Symposium on FPGAs for Custom Computing Machines FCCM'96*, March 1996.

[3] D. Hocevar and A. Gatherer. Architecture Selection for a Low Power Flexible Viterbi Decoder. In *IEEE Int. Conf. on Third Generation Wireless Communications*, San Franciso, June 2000.

[4] H. Suzuki, Y. Chang, and K. Parhi. Low-Power Bit-Serial Viterbi Decoder for 3rd Generation W-CDMA Systems. *IEEE 1999 Custom Integrated Circuits Conference*, 1999.

[5] H. Hendrix. Viterbi Decoding Techniques in the TMS320C54x Family *Texas Instruments Application Report, SPRA071*, June 1996.

[6] S.B. Wicker. Error Control Systems for Digital Communications and Storage. Prentice Hall, Englewood Cliffs, NJ, 1995.

[7] R. Cypher and C.B. Chung. Generalized traceback techniques for Survivor Memory management in the Viterbi Algorithm. In *GLOBECOM*, Dec. 1990.

[8] K. Chadha. A Reconfigurable Decoder Architecture for Wireless LAN and Cellular Systems. *M.S. Thesis*, Rice University, May 2001.