A Dynamically Reconfigurable Adaptive Viterbi Decoder

Sriram Swaminathan, Russell Tessier, Dennis Goeckel, and Wayne Burleson
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA. 01003.
{sswamina, tessier}@ecs.umass.edu

ABSTRACT

The use of error-correcting codes has proven to be an effective way to overcome data corruption in digital communication channels. Although widely-used, the most popular communications decoding algorithm, the Viterbi algorithm, requires an exponential increase in hardware complexity to achieve greater decode accuracy. In this paper, we describe the analysis and implementation of a reduced-complexity decode approach, the adaptive Viterbi algorithm (AVA). Our AVA design is implemented in reconfigurable hardware to take full advantage of algorithm parallelism and specialization. Run-time dynamic reconfiguration is used in response to changing channel noise conditions to achieve improved decoder performance. Implementation parameters for the decoder have been determined through simulation and the decoder has been implemented on a Xilinx XC4036-based PCI board. An overall decode performance improvement of 7.5X for AVA has been achieved versus algorithm implementation on a Celeron-processor based system. The use of dynamic reconfiguration leads to a 20% performance improvement over a static implementation with no loss of decode accuracy.

Keywords

Viterbi coding, FPGA, dynamic reconfiguration.

1. INTRODUCTION

Convolutional codes, which allow for efficient soft-decision decoding [9], are widely employed in wireless communication systems. As convolutional codes become more powerful, the complexity of corresponding decoders generally increases. The Viterbi algorithm [9], which is the most extensively employed decoding algorithm for convolutional codes, works well for less-complex codes, indicated by *constraint length* K. However, the algorithm's memory requirement and computation count pose a performance obstacle when decoding more powerful codes with large constraint lengths. In order to overcome this problem, the adaptive Viterbi algorithm (AVA) [4] [11] has been developed. This algorithm reduced the average number of computations required per bit of decoded information while achieving comparable bit-error rates (BER) versus Viterbi algorithm implementations.

Reconfigurable computing has been proposed for signal processing with various objectives, including high performance, flexibility, specialization, and most recently, adaptability. Reconfiguration is characterized by how fast the reconfiguration can occur and how many possible reconfigurations can be used. For many signal processing systems [13], it is possible to exploit variations in signals to vary computation and memory requirements. The recent proliferation of wireless communication systems has indicated the need to dynamically adapt communications architectures at the hardware level. These architectures can be characterized via a set of architectural parameters which can be determined experimentally.

In this paper, the analysis and implementation of an adaptive Viterbi decoder is described. The decoder implementation is the first operational implementation of the AVA algorithm and is targeted to a Xilinx XC4036XL FPGA to take advantage of computational specialization and parallelism. Our implementation is unique since it has the capability to adapt the amount of computation performed and the amount of storage used at both a fine-timescale (ms) and coarse-timescale (s) level. Reconfigurable logic is a desirable candidate for the adaptive Viterbi algorithm implementation due to the adaptive nature of the algorithm and the wide range of application parameters. Architectural parameters for the algorithm were obtained through C simulations on a workstation. In the adaptive Viterbi algorithm, the number of candidate data sequences (survivor paths) retained per received symbol (transmitted data bit) varies over time. Although reconfiguration based on each received symbol would improve performance, current FPGA technology makes this approach infeasible. A second, more effective, approach is to reconfigure AVA architecture based on channel noise characteristics. If channel noise is known to be reduced, decoder computation and memory is reduced to support faster performance.

Following implementation on a PCI-based WildOne board from Annapolis Micro Systems [2] containing a XC4036XL FPGA, a series of experiments involving the adaptive Viterbi decoder were performed. Experimental results show that the overall run-time of the decoder implementation on a XC4036XL-08 FPGA, including bus overhead, is up to 7.5 times faster than a software AVA implementation on a 366

FPGA'02, February 24-26, 2002, Monterey, California, USA.

MHz Celeron microprocessor. This performance improvement can be attributed to the parallelism available within the FPGAs. Further, it is shown that, depending on channel noise statistics, dynamic decoder reconfiguration can achieve a performance improvement of about 20% over a static AVA decoder implementation on an FPGA with no reduction of decoder accuracy. Under reduced-noise channel conditions, a less complex hardware design can be swapped into the FPGA to achieve the same decode accuracy. Due to the reduced complexity of the algorithm, logic resource requirements are reduced by more than a factor of two compared to standard Viterbi decoder implementations, even for larger constraint lengths [5] [16] and decoder performance in terms of processed bandwidth is improved. An empirical study shows that hardware requirements for our decoder grow predictably with constraint length at a rate substantially less than the exponential growth exhibited by standard Viterbi algorithms. The model is verified through experimental results.

2. BACKGROUND

Error correction coding [9] can be used to detect and correct data transmission errors in communication channels. Encoding is accomplished through the addition of redundant bits to transmitted information symbols. These redundant bits provide decoders with the capability to correct transmission errors. Convolutional codes [7] form a set of popular error-correction codes. In convolutional coding, the encoded output of a transmitter (encoder) depends not only on the set of encoder inputs received during a particular time step, but also on the set of inputs received within a previous span of K-1 time units, where K is greater than 1. The parameter K is the *constraint length* of the code.

A typical convolutional encoder of constraint length $K = 3$ is shown in Figure 1. As shown in the figure, the encoding of convolutional codes can be accomplished with shift registers and generator polynomials (XOR functions). A convolutional encoder is represented by the number of output bits per input bit (v), the the number of input bits accepted at a time (b), and the constraint length (K), leading to representation (v, b, K). Figure 1 depicts a (2, 1, 3) convolutional encoder since the encoder accepts one input bit per time step and generates two output bits. The two output bits are dependent on the present input and the previous two input bits. The constraint length K indicates the number of times each input bit has an effect on producing output bits. Larger constraint lengths, i.e. $K = 9$ or higher, are preferable since they allow for more accurate error correction. Encoding *rate* R is equal to b/v. In many communication systems, a rate of $1/2$ is used [9]. Initially, the contents of the encoder shift register are set to zero. The contents are shifted right each time a one-bit value is converted into a two-bit symbol and transmitted.

The operation of the encoder can be represented by a *state diagram*, as shown in Figure 2. Nodes represent the present state of the shift register while edges represent the output sequence and point to the next state of transition. Successive evaluation of state over time leads to the trellis diagram shown in Figure 3. The diagram is a time-ordered mapping of encoder state with each possible state represented by a point on the vertical axis. Nodes represent the present state of the shift register at specific points in time while edges represent the output sequence and point to the

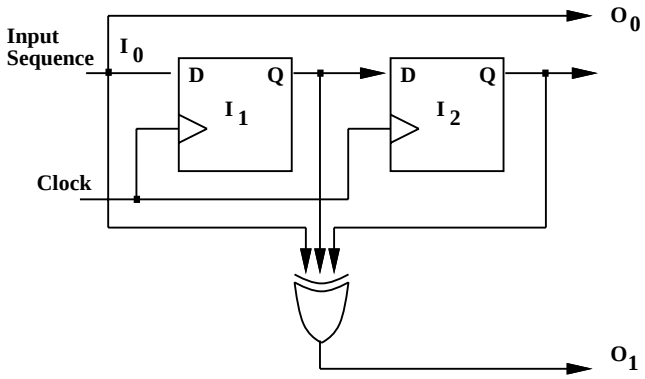

Figure 1: A (2, 1, 3) convolutional encoder

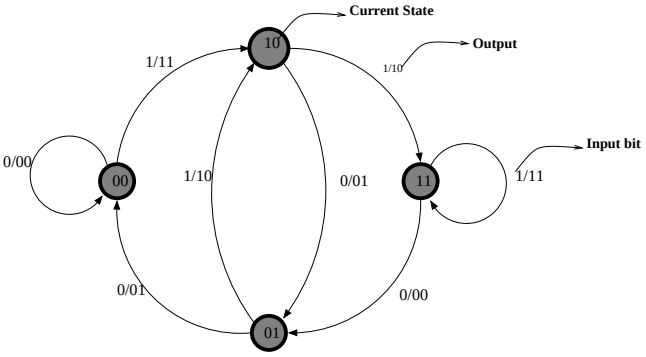

Figure 2: State diagram for the convolutional encoder in Figure 1

next state of transition. The horizontal axis represents time steps. Branch lines indicate the transition of the present state of the shift register to the next state upon receiving a particular input bit, b. The upper branch leaving a node implies an input of 0 while the lower branch implies an input of 1.

The function of the decoder is to attempt to reconstruct the input sequence transmitted by the encoder by evaluating the received channel output. Values received at the decoder may differ from values sent by the encoder due to channel noise. The interaction between states represented by the trellis diagram is used by a decoder to determine the likely transmitted data sequence [16] as v-bit symbols are received. An example received sequence of two-bit v values appears at the top of Figure 3. The cost of a particular transition edge (branch) is determined from the Hamming distance of the received symbol and the *expected* symbol, labelled in bold on the transition edge. At each node the cumulative cost or *path metric* of the path is determined. These values are labeled in bold at each node in the figure. If multiple paths converge on the same state, the lowest cost path is preserved and other paths are eliminated. After a series of time steps, referred to as the *truncation length* (TL), the lowest-cost path, also known as minimum distance path, is determined, identifying the most-likely transmitted symbol sequence. The typical value of the truncation length depends on the noise in the channel and has been empirically found to be 3-5 times the constraint length [9]. Each *path* in the trellis diagram represents a unique set of inputs,

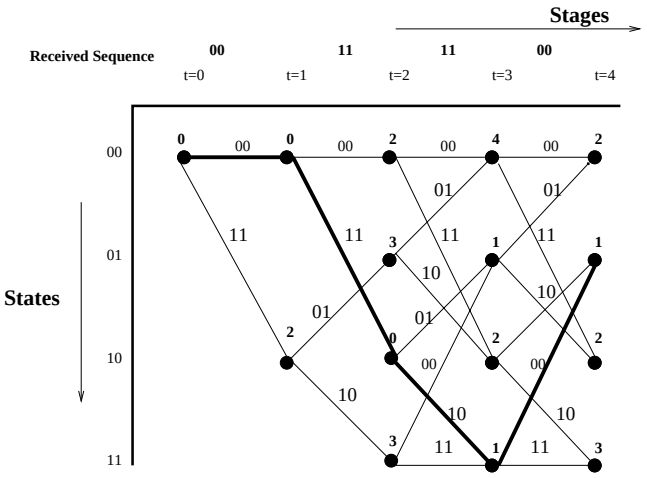

Figure 3: Trellis diagram for the convolutional encoder in Figure 1

such as the path highlighted in bold, corresponding to the lowest-cost input sequence $b = (0110)$.

The performance of a decoder is characterized by the number of decoded output bits which are in error, the *Bit Error Rate* or *BER*. The *BER* is the ratio of the number of bits in error to the total number of bits transmitted. For communication fidelity it is desirable to achieve a low BER. For this paper, a typical, maximum BER of 10^{-5} [9] is targeted. The Viterbi algorithm [9], the most popular decoding approach for convolutional codes, determines a minimum distance path with regards to Hamming distances applied to each received symbol. A limiting factor in Viterbi decoder implementations is the need to preserve candidate paths at all 2^{K-1} trellis states for each received symbol. This requirement leads to an exponential growth in the amount of computation performed and in the amount of path storage retained as constraint length K grows. Most hardware implementations of the Viterbi algorithm [9] are split into three parts: the branch metric generators (BMG), add-compare-select (ACS) units, and the survivor memory unit. A BMG unit determines Hamming distances between received and expected symbols. An ACS unit determines path costs and identifies lowest-cost paths. The survivor memory stores lowest cost bit-sequence paths based on decisions made by the ACS units.

3. RELATED WORK

Several reconfigurable implementations of Viterbi decoders have been reported. Although these systems are FPGA based, none of them use run-time reconfiguration to achieve performance improvement. In [8], a K=9 Viterbi decoder is implemented on a XC4000 FPGA. Only four add-compare-select units out of a possible 2^8 are used to compute the path metrics of all states. Unlike our approach, this implementation does not evaluate all trellis states in parallel, resulting in slower decoding operation.

In [16], Racer, a constraint length 14 Viterbi decoder, is described. The system uses 36 XC4010 FPGAs and seven processor cards and employs a novel approach to implementing survivor memory. Due to the use of a sizable number of FPGAs and significant inter-chip communication, system area is large. Although the Viterbi decoder is implemented in FPGAs, the decoder implementation remains static across varying channel data. Racer exhibits significant parallelism, although some add-compare-select hardware is multiplexed across multiple trellis states per received symbol. Candidate paths are stored in memory external to the FPGAs. Our AVA approach achieves fully parallel implementation on a single, large FPGA that contains significantly less total logic than the board used in [16] for the same constraint length (K=14).

In [5], a Viterbi decoder of constraint length 7 using four XC4028EX FPGAs is described. The decoder is partitioned so that 64 ACS units fit into two of the FPGAs and the remaining two FPGAs house the survivor memory and its corresponding controller. The main issue with this approach involves data transfer between FPGAs. Although [5] allowed for parallel trellis evaluation, the limited data rate of 12 Kbps was achieved for a relatively small constraint length of 7. This reduced rate was primarily due to inter-chip data transfer overhead. No dynamic reconfiguration was performed.

4. ADAPTIVE VITERBI ALGORITHM

The adaptive Viterbi algorithm [4] was introduced with the goal of reducing the *average* computation and path storage required by the Viterbi algorithm. Instead of computing and retaining all 2^{K-1} possible paths, only those paths which satisfy certain cost conditions are retained for each received symbol at each state node. Path retention is based on the following criteria.

1. A threshold T indicates that a path is retained if its path metric is less than $d_m + T$, where d_m is the minimum cost among all surviving paths in the previous trellis stage.

2. The total number of survivor paths per trellis stage is limited to a fixed number, N_{max}, which is pre-set prior to the start of communication.

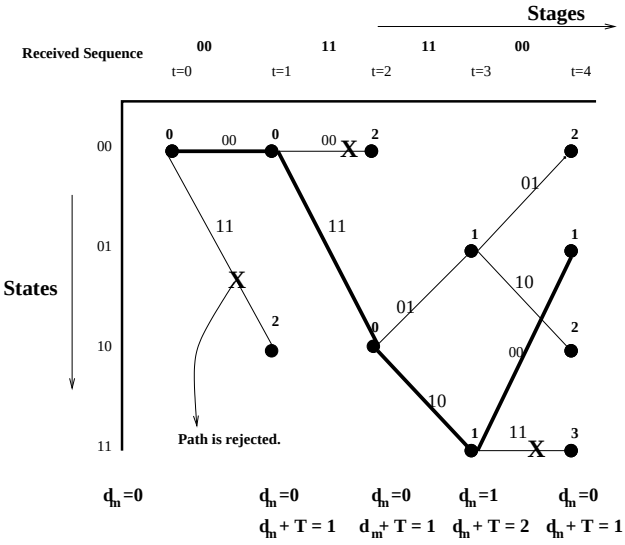

Figure 4: Trellis diagram for a hard-decision adaptive Viterbi decoder with $T = 1$ and $N_{max} = 3$

The first criterion allows high-cost paths that likely do not represent the transmitted data to be eliminated from consideration early in the decoding process. In the case of many paths with similar cost, the second criterion restricts the number of paths to N_{max}. A trellis diagram for an adaptive Viterbi algorithm of constraint length 3 is shown in Figure 4 for the same set of received symbols as shown in Figure 3. Threshold value $T = 1$ is preset for this example.

At each stage, the minimum cost (path metric) of the previous stage d_m, threshold T, and maximum survivors N_{max} are used to prune the number of surviving paths. Initially, at t=0, the decoder state is set to 00. Like the Viterbi trellis, two branches emanate from state 00 to states 00 and 10 at t=1 representing encoded transmission 0 and 1 values respectively by the encoder. If the received value at t=0 is 00, as shown in the trellis, it is more likely that a $b = 0$, $v = 00$ was transmitted rather than a $b = 1$, $v = 11$ value since both bits of the latter v would have been corrupted by noise. As a result, the *path metric* of the top branch is 0 and the bottom branch is 2. These are the Hamming distances between the received and *expected* values shown on the trellis. Since state 00 is the only state at t=0, d_m is the path metric of state 00, which is 0. As a result, $d_m + T$ is 1. At t=1, the path leading to state 10 does not survive since 2, the current path metric of state 10, is greater than 1, the value of $d_m + T$. As a result, only one branch, the branch leading to state 00 survives at t=1. The new d_m used at t=2 is the minimum among metrics of all surviving paths at t=1. Since only one path survives at t=1, d_m is 0, the path metric of state 00. At each stage the process is repeated until the truncation length is reached and the least error path can be identified.

Careful calculation of T and N_{max} is the key to effective use of the AVA algorithm. If threshold T is set to a small value, the average number of paths retained at each trellis stage will be reduced. This can result in an increased BER since the decision on the most likely path has to be taken from a reduced number of possible paths. Alternately, if a large value of T is selected, the average number of survivor paths increases and results in a reduced BER. As a result, increased decode accuracy comes at the expense of a additional computation and a larger path storage memory. Generally, the value of T should be selected so that the BER is within allowable limits while matching the resource capabilities of the hardware.

N_{max} denotes the maximum number of survivor paths to be retained at any trellis stage. The maximum per-trellis stage number of survivor paths, N_{max}, has a similar effect on BER as T. If a small value of N_{max} is chosen, paths which satisfy the threshold condition may be discarded, potentially leading to a large BER. Alternately, if N_{max} is set to a large value, extra computation and memory are required, potentially with little benefit to BER reduction. As a result, an optimal value for N_{max} should be chosen to balance hardware size and BER.

Several reduced-complexity algorithms similar to the adaptive Viterbi algorithm have been developed, although each has significant limitations. The M-algorithm [6] is a popular reduced-complexity alternative to the Viterbi algorithm. Like the AVA, complexity reduction is achieved by retaining only the best M (N_{max}) paths at each trellis stage. Unlike AVA, this approach does not use a threshold condition to determine which paths are saved but rather sorts all paths and retains the M lowest-cost paths. This requirement of sorting circuitry adds complexity and delay to the M-algorithm. Since the AVA only requires comparison to a value determined by the metric of the best extended path, sorting is not required. A *beam-search* algorithm, which is similar to the AVA, was implemented in software in [10]. This approach was used in conjunction with hidden Markov modeling (HMM) of speech.

Unlike previous AVA approaches [4], the standard operation of eliminating the largest-metric path when two survivor paths enter the same trellis state was not implemented in our approach due to hardware complexity. The implemented algorithm more closely resembles a variant of the AVA known as the Simmons T-algorithm [11].

5. AVA ARCHITECTURE

To demonstrate the benefit of the adaptive Viterbi algorithm we have developed the first hardware implementation of the algorithm. This architecture takes advantage of parallelization and specialization of hardware for specific constraint lengths and dynamic reconfiguration to adapt decoder hardware to changing channel noise characteristics.

5.1 Description of the architecture

The architecture of the implemented adaptive Viterbi decoder is shown in Figure 5 for the encoder with parameters shown in Figure 1, (2, 1, 3). The adaptive Viterbi decoder accepts two inputs from the channel which represent the outputs of the encoder that have been transmitted. The *branch metric generator* determines the difference between the received v-bit (in this case 2) value and 2^v possible expected values. This difference is the Hamming distance between the values. A total of 2^v branch metrics are determined by the branch metric generator. For v=2 these metrics are labeled b_{00}, b_{01}, b_{10} and b_{11}.

The *Add-Compare-Select* (ACS) unit, shown in detail in Figure 6, evaluates the path metric of each path and determines if paths meet AVA conditions for path survival. At each trellis stage, the minimum-value surviving path metric among all path metrics for the preceding trellis stage, d_m, is computed. New path metrics are compared to the sum $d_m + T$ to identify path metrics with excessive cost. As shown at the left of Figure 6, the path metrics for all potential next state paths, d_i, are computed by the ACS unit. Comparators are then used to determine the life of each path based on the threshold, T. If the threshold condition is not satisfied by path metric $d_m + T$, the corresponding path is discarded.

Present and next state values for the trellis are stored in two column arrays, Present state and NEXTSTATE of dimensions N_{max} and $2N_{max}$ respectively, as shown in Figure 5. There can be at the most N_{max} survivor paths at any stage. Since each path is associated with a state, the number of present states is N_{max}. Each path can potentially create two child paths before pruning as there are two possible branches for each present state based on a received 0 or 1 symbol. Entries in the NEXTSTATE array need not be in the same row as their respective source present states. In order to correlate the next state paths and next states located in the NEXTSTATE array, an array of size $2N_{max}$, called PathIdentify, is used. For each next state element, this array also indicates the corresponding row in path storage (survivor) memory for the path.

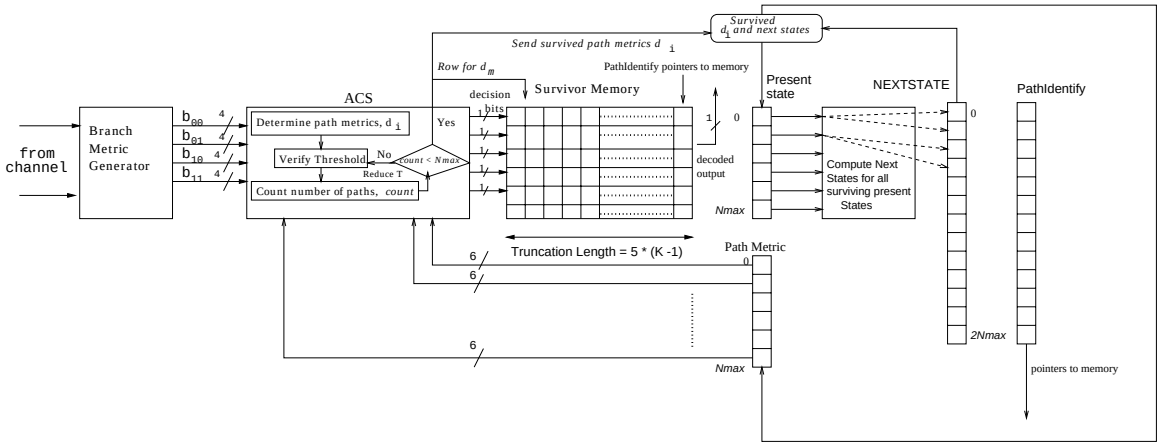

Figure 5: Adaptive Viterbi decoder architecture

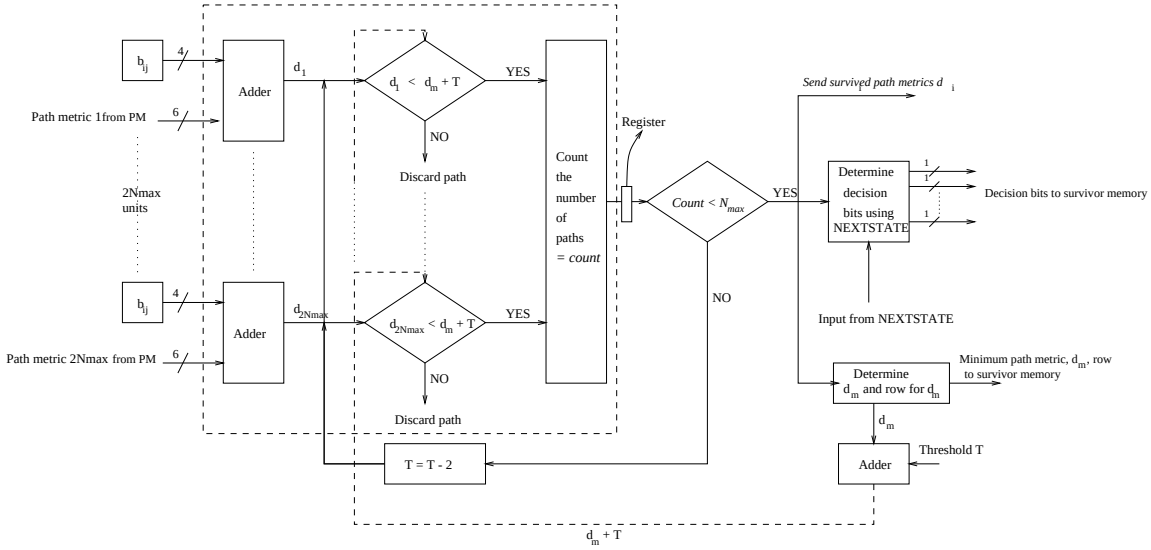

Figure 6: ACS unit of adaptive Viterbi decoder

Once the paths that meet the threshold condition are determined, the lowest-cost N_{max} paths are selected. To avoid the need for the sorting circuit described in [6] for the M-algorithm, we have developed a novel path pruning approach. Sorting circuitry is eliminated by making feedback adjustments to the parameter T. If the number of paths that survive the threshold is less than N_{max}, no sorting is required. For stages when the number of paths surviving the threshold condition is greater than N_{max}, T is iteratively reduced by 2 for the current trellis stage until the number of paths surviving the threshold condition is equal to or less than N_{max}. In Section 7 it is shown that T and N_{max} can be determined through simulation so that T reduction is needed infrequently. Following path reduction, at most N_{max} remaining trellis states are stored in the *Present state* array in preparation for the receipt of the next symbol.

The *register-exchange* based survivor memory [12] stores path sequence information and has a two dimensional size of N_{max}*TL, where TL is the truncation length. Each memory location stores an input bit, the *decision bit* from the ACS. Single-bit storage is performed for each surviving path at each trellis stage. Each row of the survivor memory is associated with a present state and has a *valid* bit to indicate the existence of a survivor path. Once survivor memory storage reaches the truncation length, the lowest-cost path sequence can be retrieved.

5.2 Architectural Model

The logic area in terms of logic blocks of our adaptive Viterbi decoder can be characterized by an empirical model in terms of parameters N_{max} and K. This expression is of the form :

$$Area = AN_{max} + BKN_{max} + C$$

where A, B, C are constant coefficients. These coefficients were determined by evaluating a set of decoders with constraint lengths between 4 and 14. Through line-fitting, coefficients for A, B, and C were determined to be 90, 5.6, and 215, respectively.

The first term in the equation accounts for logic blocks which implement path metric comparators and the registers used to store path metrics. The second term accounts

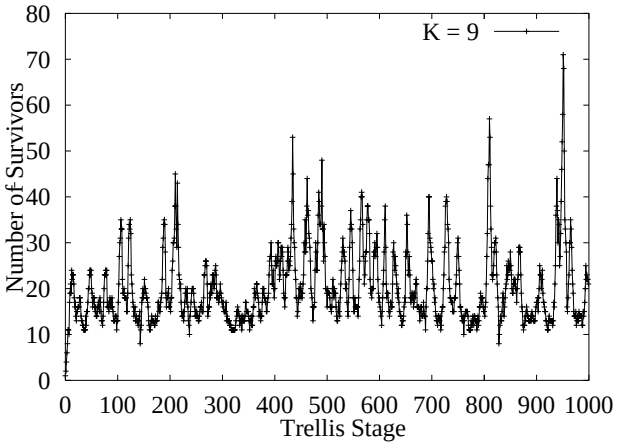

Figure 7: Number of surviving paths versus decoded symbol for the adaptive Viterbi algorithm for K = 9, r = 1/2 convolutional code with N_{max} set to 2^{K-1} (its maximum) on an un-faded channel

for path storage in memory implemented inside the FPGA. Since the truncation length (TL) is 5 * K [9], the term depends on both the maximum number of bits stored per trellis state, N_{max}, and K. Additionally, the width of present state and next state registers increases linearly with K and the number of registers increases linearly with N_{max}. The constant term in the expression accounts for logic which is fixed in size in relation to N_{max} and K. This includes the branch metric generator, which is dependent on the parameter v defined in Section 2 and the logic needed to iteratively decrement T to avoid sorting.

5.3 Suitability of Dynamic Reconfiguration

Dynamic reconfiguration of the entire adaptive Viterbi decoder hardware is considered as a means to enhance performance without compromising decode accuracy. The hardware resource requirements of an adaptive Viterbi decoder change in response to channel noise conditions for a fixed BER. Based on noise levels at a specific time instant, minimally sufficient hardware resources can be dynamically allocated to meet the BER requirements of the application while achieving maximal performance [3]. A significant amount of channel noise demands a large constraint length, such as $K = 14$, to achieve a BER similar to that achieved by a constraint length $K = 4$ for a less noisy channel. It is shown in Section 7 that decoding speed is inversely related to resource requirement. This relationship is exploited to enhance performance through the dynamic allocation of AVA logic resources. In implementing the AVA, two reconfiguration options, *fine-timescale* and *coarse-timescale* reconfiguration are considered.

Coarse-timescale reconfiguration of the adaptive Viterbi decoder, based on parameters such as K, T and N_{max}, is performed in accordance to variations in channel noise conditions over seconds. Reconfiguration at this time scale minimizes the performance impact of millisecond FPGA reconfiguration times. Coarse-timescale reconfiguration is motivated by changing channel noise characteristics from parameters such as weather, distance, or battery-power. These parameters result in a signal-to-noise ratio (SNR) that changes relatively slowly (seconds or longer). When more accurate decoding is required, a lower clock-speed decoder (larger K) can be used at the cost of reduced decode rate. When less accurate decoding is required, a higher-performance decoder is swapped in. If dynamic reconfiguration was not allowed, the lower-performance decoder would always need to be resident. Coarse-timescale reconfiguration provides an optimized but variable bit rate and is targeted at data rather than voice applications.

In *fine-timescale* reconfiguration, configuration contents are changed once every one or two decoded bits (ms) in response to the number of survivor paths retained at specific instants of time. If additional survivor paths are required, the amount of required path storage increases, potentially limiting decoder performance. If fewer survivor paths are required, a faster-performance decoder can be used instead. After experimentation it was determined that fine-timescale reconfiguration is infeasible due to rapid variations (multiple changes per ms) in retained survivor paths over time. Current FPGA architectures [1] [15] require reconfiguration times measuring milliseconds. Figure 7 shows survivor path variation for a constraint length 9 decoder.

6. EXPERIMENTAL APPROACH

To allow for parameter testing for the adaptive Viterbi algorithm, a sample data transmission system was first modeled in software. The overall communication system model that has been used for experimentation is shown in Figure 8. This system contains blocks for data generation, encoding, transmission, and decoding.

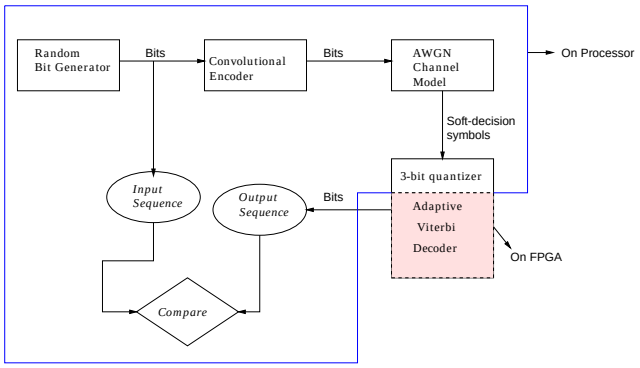

Figure 8: System model

The *Random Bit Generator* is a C module that generates a randomized bit sequence to model transmitted data. The *convolutional encoder*, also shown in Figure 1, can be parameterized to assorted constraint lengths. The encoded data from the encoder is fed to the *AWGN channel simulator*. This block simulates a noisy channel where errors are inserted into the bit sequence. The amount of noise depends on the Signal-to-Noise-Ratio (SNR) preset by the user. The channel is noisy if SNR is low. The symbols obtained from the AWGN channel model are quantized [12] before being sent to the *decoder* as its input. On receiving the input, the decoder attempts to recover the original sequence. All software modeling of the communication system was performed using a 366 MHz Celeron PC.

For hardware experimentation, the AVA design was mapped to a Xilinx XC4036XL-08 FPGA which is part of a WildOne

K	N_{max}	T	TL	CLBs	4-LUTs	3-LUTs	Flipflops
4	4	14	20	553	978	196	278
5	7	14	25	1194	2046	340	540
6	8	18	30	1206	2081	482	724
7	8	17	35	1215	2087	537	756
8	8	17	40	1284	2119	654	788
9	9	18	45	1296	2213	615	820
10	21	20	50	3371	6243	0	1911
11	25	23	55	3643	6982	0	2137
12	25	23	60	3668	7114	0	2170
14	41	24	70	6741	12876	0	2446

Table 1: FPGA resource utilization for the adaptive Viterbi decoder on XC4036XL-08 (K=4 to 9) and XCV1000-04 (K=10 to 14) for BER of 10^{-5}. Note CLB label for XCV1000 refers to CLB slices.

board from Annapolis Micro Systems [2]. This allowed for in-field testing of AVA designs for constraint lengths up to K=9 with the rest of the communication system modeled in software. To test larger AVA implementations, decoders with constraint lengths up to K=14 were mapped to a Xilinx XCV1000 FPGA, although not implemented physically in hardware. An RTL level description of the adaptive Viterbi decoder was written in VHDL that could be mapped to both XC4036 and XCV1000 devices. The VHDL code was simulated using Cadence *Affirma* tools. All designs were synthesized using Synplicity *Synplify* and mapped to Xilinx hardware using Xilinx Foundation M2.1 tools with timing constraints. The maximum frequencies of operation of the FPGAs were obtained from the Xilinx *TRACE* timing analyzer tool. For designs targeted to the WildOne board, decode rates were measured through profiling with *gprof*. For designs mapped to the XCV1000, cycle periods from TRACE were used in conjunction with cycle counts from HDL simulation to estimate decode speed.

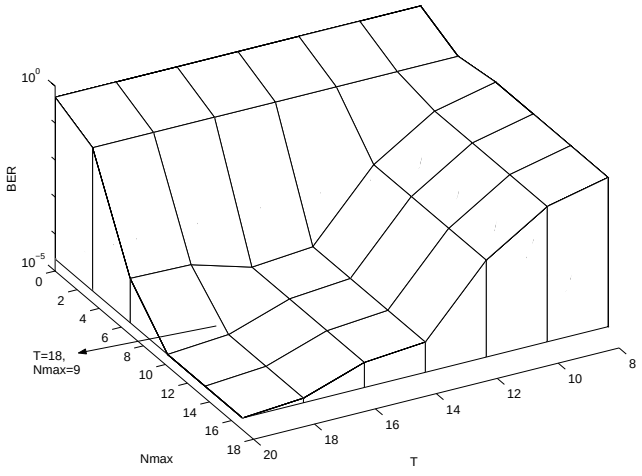

Figure 9: Variation of BER with N_{max} and T for K=9, SNR=3.1dB

K	FPGA decode (Kbps)	Decode rate w/PCI overhead (Kbps)	Max. FPGA clock (MHz)	SNR range (dB)
4	333.7	186.0	40.5	6.3-6.5
5	164.2	117.7	20.1	6.1-6.3
6	162.3	116.3	19.9	5.5-6.1
7	160.8	114.2	19.7	3.9-5.5
8	143.6	109.4	17.6	3.7-3.9
9	141.1	107.8	17.3	3.1-3.7
10	101.5	NA	25.5	3.0-3.1
12	94.8	NA	24.7	2.8-3.0
14	82.3	NA	23.0	2.5-2.8

Table 2: Decode rate versus K for XC4036XL-08 (K=4 to 9) and XCV1000-04 (K=10 to 14) for a BER of 10^{-5} and N_{max} values from Table 1.

7. EXPERIMENTAL RESULTS

7.1 Parameter Evaluation

The first set of AVA experiments determined optimal N_{max} and T parameter settings for a range of constraint lengths. Figure 9 indicates the spectrum of values possible for varying BER, T, and N_{max} for a fixed constraint length K. Similar results for a variety of constraint lengths can be found in [12]. Consider a design which requires a BER of 10^{-5}. From Figure 9, it can be seen that $T = 18$ and $N_{max} = 9$ would achieve the desired BER. Table 1 indicates representative T and N_{max} values across a range of constraint lengths for a fixed BER of 10^{-5}. The truncation length for all constraint lengths was fixed at 5 * K.

7.2 FPGA Resource Usage

The logic resources used by the adaptive Viterbi decoder architecture described in Section 5 was measured in terms of logic block (CLB) usage. Table 1 summarizes the resource utilization of the adaptive Viterbi decoder for different constraint lengths from $K = 4$ to $K = 9$ on the XC4036 FPGA and constraint lengths $K = 10$ to $K = 14$ on an XCV1000 FPGA. As shown in Table 1, an adaptive Viterbi decoder of constraint length 9 utilized 100% of 1296 XC4036 CLB resources (85% LUT utilization), while a constraint length 14 AVA decoder fits within a single XCV1000 device.

The decoding rates for the XC4036 on the WildOne board for constraint lengths $K = 4$ to 9 are given in Table 2 along with XCV1000 rates for $K = 10$ to 14. These decode rates were obtained when the FPGA was run at the maximum possible frequency. It can be noted that this frequency decreases with increasing N_{max} and K. This is a result of the critical path of the design which passes through the comparator and survivor path counter shown at the left in Figure 6. Since the complexity of the counting circuitry depends on the number of surviving paths N_{max}, performance is affected.

The decoding rates for the XC4036 constraint lengths were determined by measuring the total time taken to execute the decoder on the FPGA including 33 MHz PCI bus access time and the time to execute the *application program interface* of the WildOne board on the 366 MHz Celeron processor. The total time was then divided by the length of the sequence to obtain decode time per bit. In order to estimate the overhead of PCI bus access and WildOne API execution, raw FPGA decoding time per bit was obtained from HDL simulations with Cadence *Affirma* tools. Table 2 shows that bus and API overheads slow down decoding by a factor of 1.5 to 2.

7.3 Coarse-timescale dynamic reconfiguration

Although the adaptive Viterbi decoder can provide impressive decode rates with a static architecture implemented in reconfigurable hardware, improved performance can be achieved if the decoder is reconfigured to match required computation. In a second set of experiments, channel noise, as indicated by SNR, was used to indicate when the entire AVA architecture on the WildOne board should be reconfigured. A bit-error rate (BER) of 10^{-5} was desired at all times. Depending on the SNR, this could be accomplished with a lower constraint-length and faster decoder for high SNR and with a higher constraint-length and slower decoder for low SNR. Effective AVA architectures for each value of K were determined from the parameters shown in Table 1 for a BER of 10^{-5}.

Experiments were performed by varying the SNR of transmitted data and reconfiguring the AVA hardware based on (K, N_{max}) values that were required to achieve the desired BER. Based on the assumption that SNR can be sampled successfully every 250,000 bits [9], FPGA hardware was reconfigured up to three times for a 1,000,000 bit sequence. SNR values varied between 6.29 and 3.34dB (requiring K values between 4 and 9) and AVA configurations based on Table 1 were chosen. The overall decoding rate achieved by reconfiguring the decoder was 125.5 Kbps, including FPGA reconfiguration time of 40ms per configuration swap. This performance can be compared to a static AVA decoder of constraint length 9, the largest decoder which could fit in the XC4036. Dynamic reconfiguration leads to a 16.5% improvement over the fixed 107.8 Kbps shown in Table 2 for a BER of 10^{-5}.

In a second test of dynamic reconfiguration, a set of 10,000 SNRs were generated using a log-normal shadowing distribution [9] for a total transmission length of 2.5 billion bits. From Table 2, the decoding rate for each of the 10,000 SNRs was determined. From the set of generated SNRs, it was found that reconfiguration was performed about 7000 out of 10000 possible times. In some cases reconfiguration was unnecessary since the required decoder did not change. The total overhead for reconfiguration was (10,000 * 40ms) = 400 seconds out of a total decode time of about six hours. The resulting average decoding rate with dynamic reconfiguration was found to be 130.328 Kbps, a 20.9% improvement over a static constraint length 9 decoder with a decode rate of 107.8 Kbps. Figure 10 plots the variation in decoding speed for different K values. The average decode rate obtained with dynamic reconfiguration is shown as a dashed line.

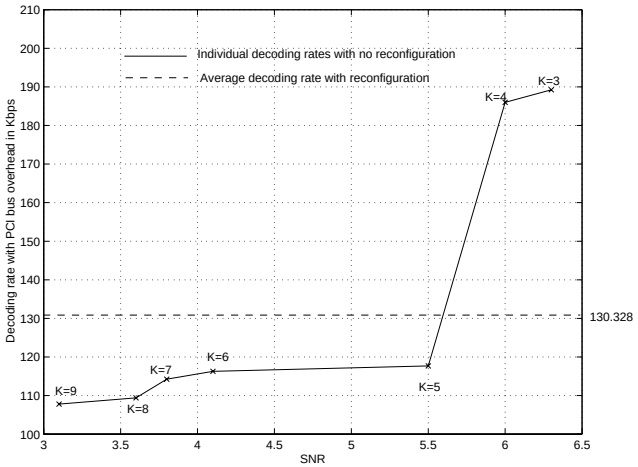

Figure 10: Effect of (K, N_{max}) based reconfiguration on decoding rate

These results are in sharp contrast with previous Viterbi implementations in FPGAs. In [16], 36 XC4010 FPGAs totaling 23940 LUTs were required to implement the Viterbi algorithm. Unlike [16], our approach allows for parallel, simultaneous evaluation of all paths. Our single-FPGA approach with dynamic reconfiguration yields a decode rate improvement of more than a factor of three versus [16] (41.1 Kbps) without any hand-optimization of FPGA synthesis or CLB placement. Our approach also can be favorably compared to [5] which used four XC4028 FPGAs and a total of 2150 CLBs. Although this Viterbi architecture allowed for parallel evaluation of paths, the decode rate of 12 kbs at a system clock of rate 1 MHz was slow for a K=7 decoder. Performance limitations compared to our single-FPGA 160.8 Kbs decode-rate approach are the result of decoder architecture including the management of the survivor path memory [12].

7.4 Comparison to a microprocessor-based implementation

In a hardware-based experiment, we compared our FPGA-based AVA implementation on the WildOne board to a software implementation running on a 366MHz Intel Celeron processor with 128 MB of memory. The software implementation was run using a variety of parameters and the decoding speeds were measured using *gprof*. Results in Table 3 and Figure 11 compare the performance of an FPGA-based adaptive Viterbi decoder to the Celeron implementation of AVA and a standard Viterbi decoder for assorted constraint lengths. Even with bus and API overheads, a speedup of about 7.5x was achieved by the FPGA implementation for constraint length 9 versus the software implementation of

K	CLBs	FFs	Decoding rate (Kbps)				FPGA Clock (MHz)	Speed up
			CPU (Viterbi)	CPU (AVA)	FPGA no overhead	FPGA PCI overhead		
4	553	278	33.3	44.4	333.7	186.0	40.5	4.2
5	1194	540	12.5	21.0	164.2	117.7	20.1	5.6
6	1206	724	4.4	19.0	162.3	116.3	19.9	6.1
7	1215	756	2.4	17.8	160.8	114.2	19.7	6.4
8	1284	788	1.2	17.2	143.6	109.4	17.6	6.3
9	1296	820	0.7	14.4	141.1	107.8	17.3	7.5

Table 3: Comparison of AVA (unless noted) decoding rates of XC4036 WildOne implementation and 366MHz Celeron processor for different Ks; FPGA was run at the maximum frequency of operation of each design based on Table 2

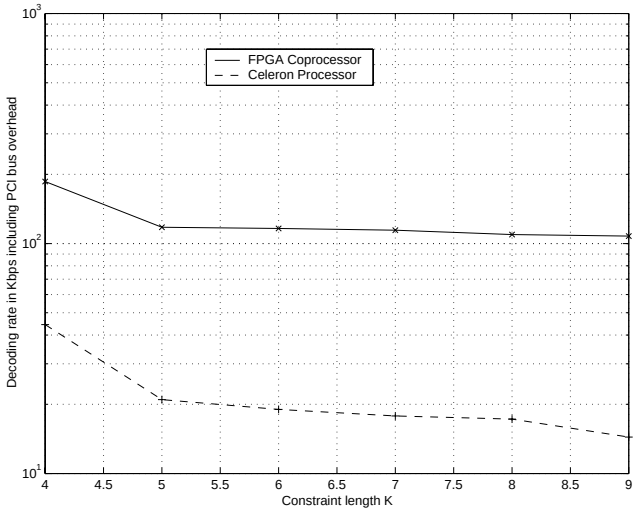

Figure 11: Comparison of decoding rates between XC4036 FPGA and Celeron processor for different Ks

AVA. A speedup of 154x was achieved by the FPGA AVA implementation versus a software version of the standard Viterbi algorithm for constraint length 9.

Figure 11 indicates that as the constraint length K and corresponding N_{max} values increase, the difference in the decoding rates between the FPGA and the processor also increases. This is due to the increasing effect of parallelism. For example, at K=3, only 6 add-compare-select units in the AVA implementation are operating in parallel, since its corresponding N_{max} is 3. However, when K=14, the corresponding N_{max} from Table 1 is 41, resulting in increased parallelism. Due to parallelism, the decoding rate does not decrease rapidly as K increases. In the case of a processor, all operations are performed sequentially. As a result, the decoding rate decreases rapidly as N_{max} increases.

7.5 Comparison to a DSP implementation

In order to compare the performance of our XC4036 FPGA-based adaptive Viterbi decoder with a DSP-based implementation, a C version of the adaptive Viterbi algorithm was compiled to a 200 MHz TMS320C6201 DSP using *Codegen* tools from Texas Instruments [14]. Subsequent simulation was performed using the stand-alone TI-DSP simulator, *load6x* [14] and cycle counts were obtained.

Table 4 shows the performance comparison between the FPGA and DSP-based implementations of the adaptive Viterbi decoder. The comparison indicates that an XC4036XL-08 FPGA implementation achieves a speed-up of up to 29x versus the DSP implementation (without bus and API overheads). The speedup decreases with increasing K and N_{max} since the maximum FPGA frequency decreases.

8. CONCLUSION AND FUTURE WORK

In this paper, a novel dynamically-reconfigurable adaptive Viterbi decoder has been presented. Important algorithm parameters have been determined through software simulation and an FPGA-based implementation has been applied to a PCI-based system. This approach has shown significant speedup versus both software implementations on a microprocessor (7.5x) and previous FPGA-based implementations (over 3x) while using a fraction of previous implementation LUT counts. Through experimentation it was shown that dynamic reconfiguration can be used effectively to improve overall performance by at least 20%. Reconfiguration was performed on the basis of channel noise to achieve a consistent bit-error rate. If channel noise increases, a more accurate but slower running decoder is swapped into the FPGA hardware. Reduced channel noise leads to the opposite effect.

We are currently exploring approaches to perform dynamic reconfiguration as a technique to reduce power consumption. As power becomes scarce, a lower-constraint length decoder with reduced decode accuracy could be used. This allows for graceful degradation of performance under power constraints, an obstacle often encountered in mobile communications. We are also considering techniques to allow for partial, rather than full, reconfiguration of the AVA decoder. Since the branch metric generator is constant across configurations, its functionality could possibly be preserved along with minimum-sized ACS circuitry that could be expanded or contracted as needed.

9. ACKNOWLEDGEMENTS

This work was sponsored by National Science Foundation grants CCR-0081405, CCR-9988238, NCR-9714597 and CCR-9875482.

10. REFERENCES

[1] Altera Corporation. *Apex II data sheet*, 2001. http://www.altera.com.

[2] Annapolis Microsystems, Inc. *WILD-ONE Reference Manual*, 1999.

K	CLBs	FFs	Decoding rate (Kbps)			FPGA Clock (MHz)	Speed up
			DSP	FPGA no overhead	FPGA PCI overhead		
4	553	278	6.301	333.743	185.994	40.455	29.5
5	1194	540	6.263	164.168	117.689	20.089	18.79
6	1206	724	6.240	162.273	116.28	19.857	18.63
7	1215	756	6.234	160.777	114.231	19.674	18.32
8	1284	788	6.222	143.632	109.392	17.576	17.58
9	1296	820	6.201	141.141	107.775	17.316	17.38

Table 4: Comparison of decoding rates of XC4036 WildOne implementation and TMS320C6201 DSP for different Ks; FPGA was run at the maximum frequency of operation of each design based on Table 2

[3] W. Burleson, R. Tessier, D. Goeckel, S. Swaminathan, P. Jain, J. Euh, S. Venkatraman, and V. Thyagarajan. Dynamically Parameterized Algorithms and Architectures to Exploit Signal Variations for Improved Performance and Reduced Power. In *IEEE Conference on Acoustics, Speech, and Signal Processing*, May 2001.

[4] F. Chan and D. Haccoun. Adaptive Viterbi Decoding of Convolutional Codes over Memoryless Channels. *IEEE Transactions on Communications*, 45(11):1389–1400, Nov. 1997.

[5] M. Kivioja, J. Isoaho, and L. Vanska. Design and Implementation of a Viterbi Decoder with FPGAs. *Journal of VLSI Signal Processing*, 21(1):5–14, May 1999.

[6] C. F. Lin and J. B. Anderson. M-algorithm Decoding of Channel Convolutional Codes. In *Proceedings, Princeton Conference of Information Science and Systems*, pages 362–366, Princeton, NJ, Mar. 1986.

[7] A. Michelson and A. Levesque. *Error-control Techniques for Digital Communication*. John Wiley and Sons, New York, NY, 1985.

[8] B. Pandita and S. K. Roy. Design and Implementation of a Viterbi Decoder Using FPGAs. In *Proceedings, IEEE International Conference on VLSI Design*, pages 611–614, Jan. 1999.

[9] J. Proakis. *Digital Communications*. McGraw-Hill, New York, NY, 1995.

[10] H. Schmit and D. Thomas. Hidden Markov Modelling and Fuzzy Controllers in FPGAs. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines*, pages 214–221, Napa, Ca, Apr. 1995.

[11] S. J. Simmons. Breath-first Trellis Decoding with Adaptive Effort. *IEEE Transactions on Communications*, 38:3–12, Jan. 1990.

[12] S. Swaminathan. An FPGA-based Adaptive Viterbi Decoder. Master's thesis, University of Massachusetts, Amherst, Department of Electrical and Computer Engineering, 2001.

[13] R. Tessier and W. Burleson. Reconfigurable Computing and Digital Signal Processing: A Survey. *Journal of VLSI Signal Processing*, 28(1):7–27, May 2001.

[14] Texas Instruments, Inc. *TMS320C6201 DSP Data Sheet*, 2001.

[15] Xilinx Corporation. *Virtex II data sheet*, 2001. http://www.xilinx.com.

[16] D. Yeh, G. Feygin, and P. Chow. RACER: A Reconfigurable Constraint-Length 14 Viterbi Decoder. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing Machines*, Napa, Ca, Apr. 1996.