A Dynamically-Reconfigurable, Power-Efficient Turbo Decoder

Jian Liang, Russell Tessier and Dennis Goeckel
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
{jliang, tessier, goeckel}@ecs.umass.edu

Abstract

The development of turbo codes has allowed for near-Shannon limit information transfer in modern communication systems. Although turbo decoding is viewed as superior to alternate decoding techniques, the circuit complexity and power consumption of turbo decoder implementations can often be prohibitive for power-constrained systems. To address these issues, we have developed a reduced-complexity turbo decoder specifically optimized for contemporary FPGA devices. Our key power-saving technique is the use of decoder run-time dynamic reconfiguration in response to variations in the channel conditions. If less favorable channel conditions are detected, a more powerful, less power-efficient decoder is swapped into the FPGA hardware to maintain a fixed bit error rate. More favorable channel conditions result in the opposite effect. Through experimentation on a Stratix-based NIOS Development Board we show that dynamic reconfiguration can result in a 52% power reduction versus a static decoder implementation. Comparisons with contemporary microprocessors illustrate a 100× performance improvement.

1 Introduction

Turbo codes [6] are widely known to provide error-correction capability within 1 dB of the Shannon limit. A limiting aspect of turbo code usage is the complexity of its hardware implementation. Typical implementations often require 2× to 5× more arithmetic operations [5, 16] than competing Viterbi techniques. Additionally, the parameters associated with turbo coding in a wireless environment (signal-to-noise ratio, available power, etc.) can frequently change, requiring a hardware implementation that offers both flexibility and parallelism. For many mobile communication systems, power and performance are the most important system design issues. Although FPGAs are often considered to be power inefficient compared to their ASIC counterparts, the reconfiguration capabilities offered by FPGA devices provide an opportunity to match application power consumption to existent operating parameters. Contemporary microprocessors and DSPs provide run-time power flexibility for turbo decoders via voltage scaling [12, 16], but the functional parallelism present in these devices is likely to be insufficient for computation as the need for decoder accuracy increases.

To address this need we have developed an FPGA-based turbo decoder architecture which matches the decoder architecture to available FPGA resources. Power reduction is achieved through the use of run-time decoder reconfiguration in response to changing channel conditions. Our *Adaptive Soft Output Viterbi Architecture (ASOVA)* turbo decoder has been derived from an existing SOVA decoder [10] and provides reduced computational complexity and a competitive bit error rate (BER) for decoders with the same operating parameters. Computationally-expensive on-chip memory updates have been customized to match the on-chip memory structures of contemporary FPGAs. The architecture and power-consumption of the decoder are dynamically varied in response to channel signal-to-noise (SNR) variations while maintaining a fixed BER. Dynamic reconfiguration is used to ensure that the lowest-power decoder that meets the required BER is present in the FPGA.

The benefits of our decoding approach have been verified via experimentation. Following simulation to determine decoder parameters, a series of ASOVA turbo decoders were designed and mapped to an Altera Stratix FPGA on a NIOS Development Board [2]. Experimental results show that by using dynamic reconfiguration, the ASOVA decoder consumes 52% less power when compared to a static decoder while providing a fixed BER. Performance comparisons for a software version of ASOVA using an NIOS microprocessor [2] and a Pentium IV showed a performance improvement of over 100×. A power reduction of 30% is achieved for the hardware decoder versus a NIOS software implementation.

The rest of the paper is organized as follows. Section 2 introduces turbo codes and the existing SOVA decoding algorithm. Our new ASOVA decoding algorithm and associated parameter exploration is described in Section 3. Section 4 presents the architecture of our ASOVA turbo decoder and the tradeoffs required for FPGA implementation. The experimental approach used for simulation and hardware test are described in Section 5 and experimental results and analysis are provided in Section 6. In Section 7, we describe related work. Section 8 summarizes our efforts and offers directions for future work.

2 Background

2.1 Turbo Codes

Turbo codes [6] are error-correction codes that rely upon redundant data transmission. By adding parity bits to transmitted data, turbo codes allow the receiver to correct errors caused by the channel. The architecture of a turbo encoder at the transmitter influences the decoder architecture at the receiver. The most important parts of a typical turbo encoder are *component encoders* which accept input data and generate encoded symbols. As shown in Figure 1, a component encoder consists of a shift register augmented with *generator* functions (AND and XOR gates). Each input bit u_k is augmented with one or more parity bits p_k. The parameter input vectors to the generator AND gates are often specified as octal values (or *codes*). For example, if (g_{10},g_{11},g_{12} = 1,1,1) and (g_{20},g_{21},g_{22} = 1,0,1) in Figure 1, a (7, 5) code-based encoder is specified. In our work, previously-determined codes with high performance, $(15,13)$, $(31,27)$, and $(65,57)$ are used [13, 18]. The constraint length K for the encoder indicates the number of times each input bit in the shift chain influences a parity output. For the example in Figure 1, K is 3. Note that the number of flip flops in the encoder can be deduced from both K and the generator code.

In general, turbo encoders contain two component encoders. For the turbo encoders considered in this work, both component encoders generate a single parity bit, resulting in the transmitted values (u, p_1, and p_2) for each encoder input bit (*code rate* 1/3). The data stored in the flip flops of a component encoder can be regarded as an encoder **state**, with state changes defined via a state machine which starts from state 0. The state machine inputs a bit u_k and outputs the parity bit p_k together with u_k at each time step.

After u, p_1, and p_2 are sent through a transmission channel, they ultimately arrive at the turbo decoder as received values y, p and q. Due to channel noise, the received values may not match their transmitted counterparts. The turbo decoder attempts to reconstruct transmitted u values through a series of decoding steps. As shown in Figure 2, a typical

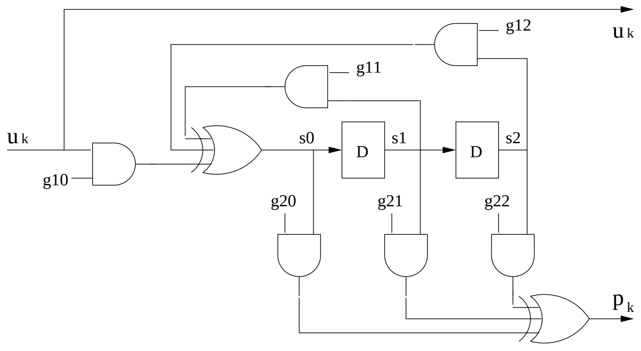


Figure 1. Component encoder architecture

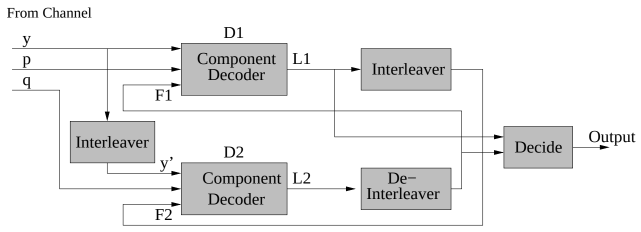


Figure 2. Turbo decoder architecture

turbo decoder [6] consists of two identical component decoders, *D1* and *D2*, interleaver/de-interleaver blocks, and an output decision block. The interleavers permute the data bits to support the error correction algorithm. The output from one decoder is fed into the other through an interleaver/de-interleaver to help the latter decoder make a better decoding decision in subsequent decoding iterations. Multiple iterations are required before the decoder converges to a final result. After a pre-specified number of decoding iterations, the final decision is made in the *Decide* block by combining the outputs from both decoders.

Several algorithms have been implemented for the *component decoder* in turbo decoders. MAP [4] and Log-MAP [13] algorithms are optimal with respect to bit error probability but exhibit significant computational complexity due to extensive searching and memory accesses. The Max-Log-MAP algorithm [9] reduces complexity by replacing logarithm computations with comparisons. The Soft-Output Viterbi Algorithm (SOVA) [13] significantly reduces complexity over other approaches by limiting searching and overall memory accesses. For a BER of 10^{-4}, the performance of SOVA is approximately 0.7 dB worse than that of MAP [13]. We choose to base our decoder on SOVA because it requires half the computation of Max-Log-MAP [23] and maintains competitive bit error performance.

2.2 Soft Output Viterbi Algorithm

Although a full description of the SOVA algorithm is beyond the scope of this paper (interested readers may consider [13]), we provide the algorithm basics necessary to understand our turbo decoder architecture. Like better known Viterbi algorithm decoders [8], a SOVA decoder determines a corrected output bit sequence by using an approximation of the encoder state, shown in the flip flops in Figure 1, and received channel values. To distinguish decoder-approximated values from encoder variables (u_k, p_k), the symbols $(\hat{u}_k, \hat{p}_k)$ are used. 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 state represented by a point on the vertical axis (note $K = 3$ for this trellis). The horizontal axis represents time steps. Each edge emanating from each state node indicates a specific encoder bit u_k for the state and leads to the *next state* to be held in the flip flops. The edge leaving a state node represents a u_k value of 0 or 1. A more detailed discussion of a similar trellis diagram can be found in [25]. A cost, called a *branch metric*, is associated with each trellis edge. For a turbo decoder, this cost represents the likelihood that the decoder inputs y and p were generated by the specific u_k indicated by the branch. For turbo decoders, a high cost metric represents a close match.

To reconstruct a sequence of received u_k values, a path of multiple trellis stages can be followed in the trellis diagram. The accumulated branch metrics along a path form the *path metric* for a specific terminal state node and represent the likelihood that a sequence of received bits matches bits transmitted by the decoder. The concepts of branch and path metrics are illustrated by the trellis in Figure 3. The path metric of each state appears above the state in the figure. When multiple paths converge into the same node, the path with the maximum path metric is retained and its path metric is marked on the node.

For SOVA-based turbo decoder implementations, the completion of a time step requires a **write** phase to store path metrics and $\hat{u}_k$ bits associated with path trellis edges to path storage memory. After a series of steps, referred to as the *truncation length* (TL), the path with the largest path metric is determined, identifying the Maximum Likelihood (*ML*) path and its associated bit sequence $\hat{u}_k^{ML}$ is the decoded output. A typical value of the truncation length is five times the constraint length [21]. In Figure 3, the *ML* path is shown with dashed lines. In SOVA, the path metric represents the likelihood that a path is the decoded path, and a larger metric implies an increased likelihood. The decoded bit sequence $\hat{\mathbf{u}}$ is obtained by a **traceback** along the TL edges of the *ML* path for the bits $\hat{u}_k^{ML}$ associated with the edges.

In addition to obtaining the decoded bit sequence, the SOVA algorithm determines reliability information Δ (also called soft output) for each decoded bit. Each Δ is determined by calculating the *difference* between the two path metrics that converge at each state node. Note that the surviving path metric becomes the path metric for the code while the lower-metric path (also called the competitive path) is eliminated. In most SOVA implementations (including ours), branch metric, path metric, and soft output (Δ) values are stored as two's complement integer values. In Figure 3, Δ is equal to (2.5 - 1.0 - branch metric for path from state 0 to state 2). The competitive path is shown with dotted lines.

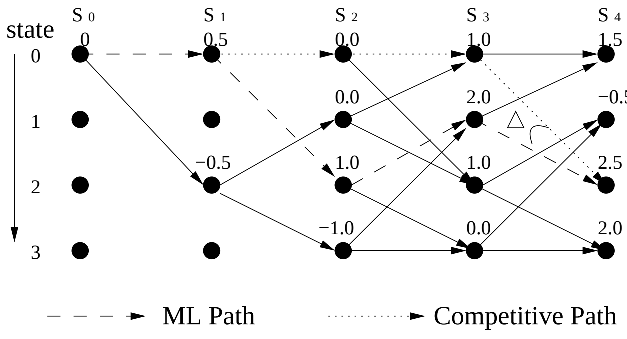


Figure 3. Trellis for SOVA component decoder

An important phase of the SOVA algorithm is the dynamic **update** of Δ in earlier trellis stages as later stages are reached [13]. As shown in Figure 4, after tracing back TL stages from trellis state i, the next U stages are checked for Δ update via the following equation.

$$\Delta_t = \min_{\hat{u}_{ML(t)} \neq \hat{u}_{cmp(t)}} (\Delta_{i-TL}, \Delta_t), t = i - TL, ..., i - TL - U \quad (1)$$

where t is the stage index, $\hat{u}_{ML}$ and $\hat{u}_{cmp}$ are the decoded bits of the *ML* path and the competitive path at a trellis stage, and U is a parameter. Through experimentation, we confirmed that $U = \frac{1}{2}TL$ [15] provides the best decode result.

3 Adaptive Soft-Output Viterbi Algorithm

Although a SOVA component decoder exhibits lower computational complexity than competing MAP, Log-MAP, and Max-Log-Map approaches, the total number of path metric evaluations per state for SOVA is still proportional to 2^{K-1}. To facilitate the subsequent implementation of turbo decoders in FPGAs, we consider new algorithmic techniques to reduce the complexity of SOVA while preserving the BER of the resulting decoded bit stream. Rather than preserving all 2^{K-1} state path metrics for each trellis stage, this new adaptive SOVA (ASOVA) approach attempts

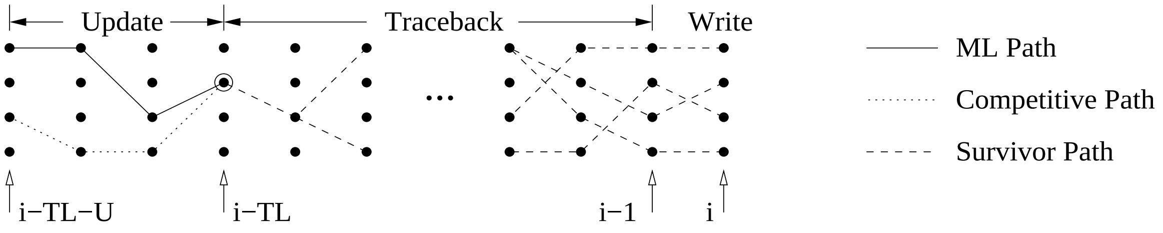


Figure 4. Traceback and update SOVA phases

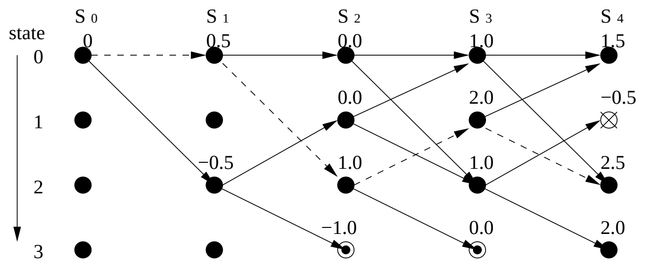


Figure 5. Trellis diagram for an ASOVA component decoder ($T = -2.0, N_{max} = 3$)

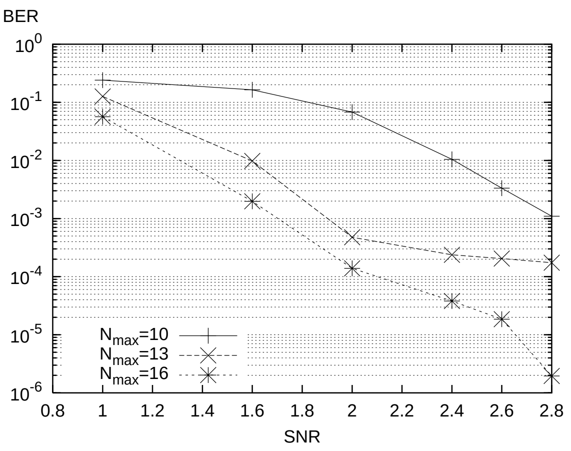


Figure 6. BER performance versus SNR for ASOVA decoders using varying N_{max} and $T = -10$

to eliminate intermediate trellis paths during processing that are least likely to lead to the decoded output bit sequence. Like the Simmons T-algorithm [24] variant of the Viterbi algorithm, path reduction for ASOVA is accomplished by considering the following criteria in evaluating path metrics.

- A threshold T (≤ 0) is used to evaluate path metrics. If the path metric of a path is less than $d_m + T$, where d_m is the maximum path metric of the previous trellis stage, the path is discarded.

- A total of at most N_{max} paths are retained at each trellis stage. If more than N_{max} paths survive path metric pruning with T, the N_{max} paths with the highest path metrics are retained.

Since it is unlikely for a low metric path to become the *ML* path later, path pruning has a low likelihood of changing the *ML* path. As a result, ASOVA generates the correct decoding sequence $\hat{\mathbf{u}}$ with a smaller number of metric paths, reducing the decoding complexity and memory usage. An example is given in Figure 5, where the numbers on top of each node represents paths metrics, $T = -2.0$, and $N_{max} = 3$. The node obstructed by a cross represents a path pruned using threshold T. The nodes encircled by an open circle indicate paths pruned using N_{max}. For example, consider the bottom node in stage S_2. The maximum path metric in stage S_1 is 0.5, so d_m is set to this value and $d_m + T = -1.5$. Since all path metrics in stage S_2 are greater than -1.5, the state with the smallest path metric is pruned so that only $N_{max} = 3$ survivors remain. In stage S_4, the path metric for state 1 (-0.5) is less than the d_m value for stage S_3 (2.0) plus -2.0 so the path is pruned due to T.

The BER versus *SNR* performance of a turbo decoder using ASOVA for several N_{max} values is shown in Figure 6. The experiment uses a (31,27) code with blocks of 1024 data bits. It can be seen that more than 0.8 dB will be lost at a BER of 10^{-4} when the survivor number is restricted to $N_{max} = 13$.

3.1 ASOVA Algorithm Enhancements

Like SOVA, the soft output Δ of ASOVA is generated by the metric difference between the *ML* path and its competitor path. When the paths with low path metrics are pruned, it is likely that competitors will also be discarded since they often have similar metrics. The loss of competitors can result in inaccurate soft outputs.

To compensate for this potential loss, two new algorithmic techniques were developed. The first approach determined an *expected value* for soft outputs along intermediate paths that have been pruned. The second approach involves

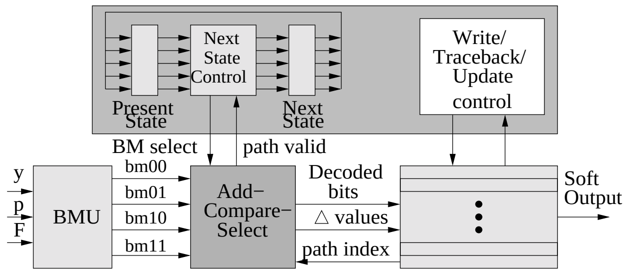


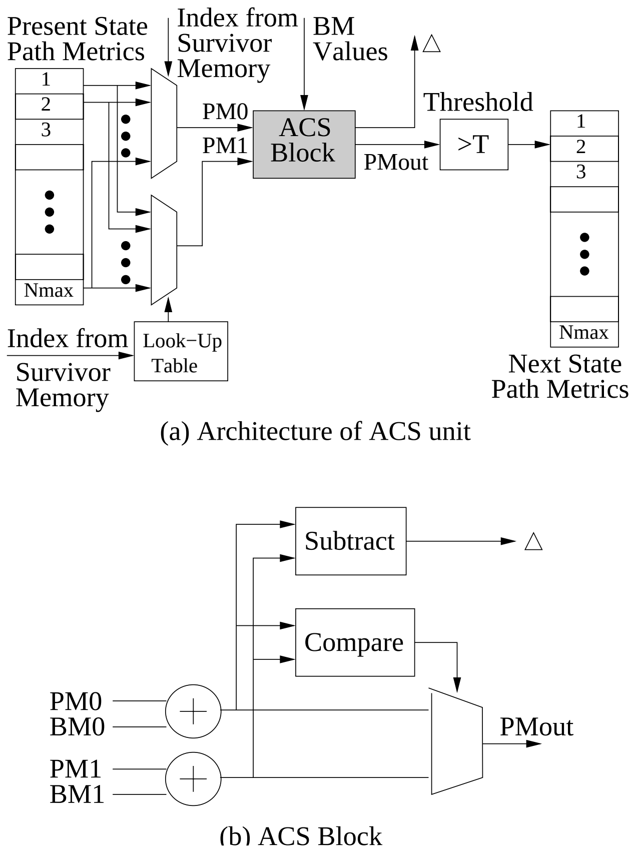
Figure 7. Component decoder architecture

the application of a scaling factor $\alpha < 1$ to ASOVA soft output values. As mentioned in Section 2.2, the soft output Δ is the path metric difference at the state node with the highest path metric. The path difference is created by subtracting the path metrics of the paths that converge on the state, the *ML* path and the competitive path. In cases when the competitive path for an *ML* output has been pruned, an *estimation* of the path metric can be used in its place. For ASOVA, this value was determined through analysis to be $\frac{2}{\sigma^2}$ [17], where σ is the standard deviation of the transmission channel noise. Since a number of intermediate competitive paths may be pruned during ASOVA processing and repeated use of estimated path metrics may be necessary, resulting soft outputs for ASOVA are often larger than the corresponding values determined using SOVA. Through experimentation we have determined that this increase can be characterized via a fixed *scaling factor* $\alpha < 1$. Multiplication of each soft output by α compensates for the use of estimation.

4 ASOVA Hardware Architecture

A turbo decoder based on our ASOVA component decoder architecture was developed and tested in an Altera Stratix FPGA. The basic components of the decoder follow the blocks shown in Figure 2. To allow for pipelined decoding, the interleaver and de-interleaver each include two data buffers. While one buffer receives new data, stored data can be read from the other. An entire 1024 bit code word can be stored in each buffer.

The component decoder is the key unit of the turbo decoder. As shown in Figure 7, the component decoder consists of four parts: the branch metric unit (BMU), the add-compare-select (ACS) unit, the survivor memory, and the control path. BMU and ACS are used to generate the new path metrics for each trellis stage. The survivor memory stores decoded bit u_k and soft output Δ_k values and performs **write**, **traceback**, and **update** operations. The control path determines next state values and controls data flow between the other three units. Detailed architectural descriptions are provided in the following sections.



(a) Architecture of ACS unit

(b) ACS Block

Figure 8. Add-Compare-Select components

4.1 Branch Metric Unit

The BMU generates the branch metrics for all four possible encoder output pairs $\hat{u}_k, \hat{p}_k$. Received decoder input values y and p and the soft output feedback F from the alternate component decoder, (e.g. $F1$ and $F2$ in Figure 2) are used to generate the branch metric for each $\hat{u}_k, \hat{p}_k$ combination. For soft output decoders, channel values y and p are quantized to multi-bit values while $\hat{u}_k$ and $\hat{p}_k$ are single bits. For ASOVA, the branch metric of a given $\hat{u}_k$,$\hat{p}_k$ at trellis stage k with a soft output feedback F is:

$$bm(\hat{u}_k \hat{p}_k) = \frac{1}{2}\hat{u}_k F + \frac{L_c}{2}(y\hat{u}_k + p\hat{p}_k) \qquad (2)$$

where L_c is equivalent to $\frac{2}{\sigma^2}$, and σ is the standard deviation of the transmission channel noise. Since $\hat{u}_k$ and $\hat{p}_k$ are binary numbers, each BMU output requires two adders and a multiplier. The DSP blocks inside the Altera Stratix FPGA are used for multiplication.

4.2 Add-Compare-Select Unit

The goal of the add-compare-select unit is to add the branch metric for a trellis edge to the path metric of the present trellis state to create a new path metric for a *next state* in the next trellis stage. This metric is then compared to the computed path metric from the competitive path to determine the survivor for the next state. The ACS operation must be performed for at most N_{max} present state path metrics for each trellis stage.

The hardware architecture used to perform this ACS computation for each path is shown in Figure 8a. The correct index into the present state path metric array is obtained from the survivor memory and is used to select path metric $PM0$. This index is also used as an input to a preprogrammed look-up table to select the path metric $PM1$ for the competitive path. As shown in Figure 8b, the branch metric $BM0$, $BM1$ for each path is added to the appropriate path metric to create new path metrics for the next state. A subtractor takes the difference of the new metrics to generate the needed soft output Δ value for the next state. A comparator selects the largest of the two path metrics $PMout$ for survival.

As shown in Figure 8, the ACS unit employs the threshold $d_m + T$ to prune low cost paths. Only those paths whose metrics fulfill the threshold requirement are subsequently stored in the next state path metric array. The array index used to store the next state path metric is stored in the survivor memory. If more than N_{max} paths survive, the threshold T is dynamically increased and ACS computation is reperformed with the new T.

4.3 Survivor Memory Unit

The survivor memory is a two dimensional memory array with N_{max} rows and $2{*}TL$ columns. The memory uses *traceback* pointers [8] so that data movement is limited. Each word in the survivor memory stores the decoded bit $\hat{u}_k$, metric difference Δ, and pointers to the previous trellis stage survivor memory values along the saved and competitive paths. As described in Section 2.2, the survivor memory supports three operations for up to N_{max} trellis states for each decoded input value: **write**, **traceback** and **update**. For a single trellis stage, a memory **write** requires a write port, **traceback** requires a read port, and the **update** phase requires two read ports to read ML and competitive path Δ values, and a write port for new Δ values. As a result, the survivor memory requires a total of 2 write ports and 3 read ports.

To facilitate FPGA implementation, our ASOVA memory is partitioned into banks. As shown in Figure 4, survivor memory **traceback** and **write** operations occur in portions of the memory that are isolated from the **update** phase. As a result, the survivor memory is partitioned vertically into eight separate banks. To save power, four banks store the two path indices and $\hat{u}_k$ values and the other four banks store the Δ values. The memory can be implemented using general two-port RAM blocks. A single memory read and write operation is performed in one clock cycle.

5 Experimental Approach

5.1 Test Platform

To test the practicality of our reconfigurable ASOVA-based architecture, a hardware implementation of the decoder was tested as part of a communication system. This system contains blocks for data generation, encoding, transmission, and decoding. A random bit *generator* creates a bit sequence to model transmitted data. A turbo *encoder*, also shown in Figure 1, then encodes the data for transmission. A *modulator* converts the coded bits into real numbers: 0 -> 1, 1 -> -1 for the binary phase-shift keyed (BPSK) system employed. The output of the modulator is input to a *AWGN channel simulator*. This block simulates a noisy channel where white Gaussian noise is added to the transmitted signal. The amount of noise depends on the signal-to-noise ratio preset by the user. The symbols obtained from the AWGN channel model are quantized 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 (except for the FPGA-based decoder) was performed using a 1.6 GHz Pentium IV PC.

5.2 Hardware Implementation

The ASOVA-based decoder architecture was mapped to a Stratix EP1S10 FPGA located on an Altera NIOS Development Board [2]. This mapping allowed for in-field testing of turbo decoder designs for constraint lengths up to K=6. An RTL level description of the turbo decoder was written in Verilog so that it could be mapped to FPGA devices. The Verilog code was simulated using Altera Quartus II simulation tools. All designs were synthesized and mapped using Quartus II with timing constraints. The maximum operating frequencies of the FPGA were obtained following Quartus II compilation. Overall communication system decode rates were measured through profiling with the *time* utility on the PC.

Power consumption values for the turbo decoders were determined using the Quartus II power analyzer. To account for power consumption during EP1S10 reconfiguration, the power associated with reading the configuration bitstream from SDRAM and storing it in the FPGA was calculated. It was determined that approximately 125 mW of power are needed during reconfiguration to read the 3,534,640 EP1S10 configuration bits from 4M$\times$32 Micron MT48LC4M32B2 SDRAM [19]. This value was determined by scaling the specified maximum power dissipa-

K	N_{max}	SNR (dB)	LUTs	MEM (Kbit)	FFs
6	32	0-1.5	4611	135.0	2523
6	28	1.5-2.0	4407	133.0	2347
6	18	2.0-2.5	3814	126.0	1907
6	12	2.5-3.0	3100	65.0	1503
6	9	3.0-4.0	2851	63.1	1371
6	7	4.0-4.5	2524	37.2	1143
6	6	4.5-5.5	2406	36.6	1099
6	5	5.5-6.0	2317	36.0	1055
6	4	>6.0	1972	25.1	871
5	16	0-1.5	2809	67.6	1293
5	12	1.5-2.0	2587	65.0	1133
5	9	2.0-2.5	2392	63.1	1013
5	8	2.5-3.0	2202	37.9	897
5	6	3.0-4.0	2074	36.6	817
5	5	4.0-5.0	1996	26.0	777
5	4	5.0-6.5	1768	25.1	661
5	3	>6.5	1722	24.4	621
4	7	0-2.0	1896	26.8	687
4	6	2.0-2.5	1831	26.5	651
4	5	2.5-4.0	1752	26.2	615
4	4	4.0-5.5	1563	20.7	535
4	3	>5.5	1541	20.4	499

Table 1. ASOVA decoder statistics for BER=10^{-4} and T=-10

tion at 200 MHz by the required FPGA configuration speed. Negligible power (5 mW) is consumed by the SDRAM after the FPGA is configured. The amount of power required to reconfigure the EP1S10 was approximated by assuming the use of an on-chip reconfiguration shift chain. The power dissipated by the shift chain was determined by calculating the energy dissipated by a single shift in 0.13 μm technology with SPICE. This shift chain energy value was scaled by the required 3,534,640 shifts and divided by configuration time to calculate FPGA reconfiguration power. It was calculated that 54.8 mW are required to reprogram the configuration bits of the EP1S10. Total EP1S10 reconfiguration time is 35 ms [3].

6 Experimental Results

6.1 ASOVA Parameter Evaluation

Prior to implementing the ASOVA component decoder in hardware, a set of simulations were performed to evaluate appropriate T and N_{max} values for our ASOVA-based decoders. Via simulation it was determined that a $T = -10$ value and the associated N_{max} values shown in Table 1 were best suited for our decoders for a fixed BER of 10^{-4}. The signal-to-noise ratio (SNR) range supported by each tested decoder is shown in Table 1. For a constraint length $K = 4$, code (15, 13) was used, for $K = 5$, code (31,27) was used, and for $K = 6$, code (65,57) was used. When the SNR is high, a reduced N_{max} can be used to obtain the required BER.

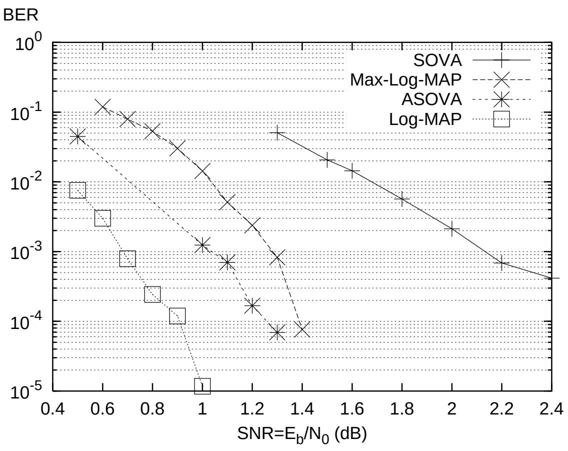


Figure 9. ASOVA performance for a (31,27) code versus competing decoder algorithms

Parameter values were used to evaluate the ASOVA decoders error-correcting performance versus competing component decoders. For comparison purposes, software versions of turbo decoders based on SOVA, Log-MAP, and Max-Log-MAP component decoders were also written. Figure 9 indicates that the BER performance of ASOVA is superior to SOVA and approaches the performance of the computationally more expensive Log-MAP algorithm for the (31,27) code ($K = 5$). Other codes demonstrated similar results.

6.2 ASOVA Turbo Decoder Implementation

To test the power consumption and decoding speed of our ASOVA-based turbo decoders, a parameterizable decoder was written in Verilog. Decoders for a variety of K and N_{max} values were synthesized to an Altera Stratix EP1S10 FPGA and downloaded to a NIOS Development Board [2]. In the following experiments, turbo decoders were tested with 1024 bit data blocks and 6 decode iterations. The survivor memory was constructed from eight memory banks with the capacity to hold $10 \times K$ trellis columns of $\hat{u}_k$ and Δ information ($5 \times K$ for **traceback**, $2.5 \times K$ for **update**, $2.5 \times K$ for rotating spare storage).

Table 1 illustrates the hardware resource usage of the decoders. Table 2 shows the decode rate and power consump-

		50 MHz		Max speed		
K	N_{max}	Power (mW)	Speed (Kbps)	f_{max} (MHz)	Power (mW)	Speed (Kbps)
6	32	447.7	173.4	52.9	469.1	183.4
6	28	431.3	193.6	56.7	485.1	219.4
6	18	306.9	228.2	59.3	431.3	270.5
6	12	232.6	288.6	60.0	279.6	346.5
6	9	212.8	410.9	67.1	292.9	551.4
6	7	177.9	447.1	66.1	233.0	590.2
6	6	173.8	450.2	68.7	228.4	619.5
6	5	168.4	501.3	67.4	215.8	677.9
6	4	147.3	469.2	77.8	159.1	728.6
5	16	205.8	312.2	70.5	280.5	440.0
5	12	193.0	301.0	70.4	250.4	424.0
5	9	148.3	411.3	73.7	206.1	606.4
5	8	169.6	444.8	74.6	246.5	663.5
5	6	130.1	468.6	79.7	181.9	746.9
5	5	134.3	470.1	80.7	198.9	758.5
5	4	110.4	472.6	82.8	163.4	782.2
5	3	100.1	471.7	82.3	143.4	776.2
4	7	134.3	487.8	81.2	248.9	792.5
4	6	125.9	515.6	81.8	204.1	851.2
4	5	113.0	622.8	80.2	198.0	851.2
4	4	106.7	688.2	84.1	176.5	1216.0
4	3	98.6	734.6	89.0	185.2	1178.0

Table 2. ASOVA performance on a Stratix EP1S10 FPGA

tion of the decoders for a range of K and N_{max} values. Two sets of power and decode rate values were determined: values at 50 MHz, the clock speed of the NIOS board, and values for the maximum possible clock rate for the decoder. Clock speeds of nearly 90 MHz were found for smaller decoders. For a 50 MHz decoder it can seen from the table that for $K = 5$, a 51% power reduction takes place across N_{max} values of 16 to 3, and for $K = 6$, a 67% power reduction takes place across N_{max} values from 32 to 4.

6.3 ASOVA Dynamic Reconfiguration

A second set of experiments were used to determine power savings that could be achieved if the entire FPGA decoder was reconfigured at run-time to support existent channel SNR requirements. Depending on the SNR, power savings are achieved by using a lower N_{max}, lower-power decoder for high SNR and a higher N_{max}, higher-power decoder for low SNR. The three constraint lengths K offered three separate SNR ranges for testing.

A set of 10,000 SNR values were generated for each K

K	Avg. Speed (Kbps)	Reconfigs required	Avg. Power (mw)
6	359.1	8369/10000	216.2
5	429.4	6306/10000	131.7
4	598.6	6925/10000	111.6

Table 3. Dynamic reconfiguration statistics

using a log-normal shadowing distribution [22] for a total transmission length of 2.5 billion bits. Based on the assumption that SNR can be sampled successfully every 250K bits [21], the FPGA was periodically reconfigured during the transmission process. Table 3 shows the number of required reconfigurations, the resulting decode rates at 50 MHz, and the average power dissipated. The average power consumption for the (31,27) code ($K = 5$) is 131 mW, a 36% improvement over a fixed $N_{max} = 16$ decoder. For a (65,57) code ($K = 6$), the average power of 216 mW is 52% less than the power of the fixed $N_{max} = 32$ decoder, 448 mW. Power and decode rate numbers include the time and power needed for FPGA reconfiguration and the time and power needed to read associated configuration bits from SDRAM, as described in Section 5.

Derived power numbers are based on common standards-based assumptions [20] regarding communication systems. The circuitry required to determine existent SNR and associated decoder K and N_{max} values is assumed to be located external to the decoder in a power-rich operating environment. Upon detection of an SNR change, this circuitry sends new K and N_{max} values to the decoder to initiate FPGA reconfiguration.

6.4 Performance Comparison to Microprocessor Implementations

Although the parallelism and memory structure of turbo decoders make efficient implementation on a microprocessor difficult, we contrasted the software performance of ASOVA on two microprocessors versus FPGA hardware implementations. Software results were determined using the 1.6 GHz Pentium IV PC (the host for the NIOS board) and the 50 MHz NIOS processor running on the FPGA board. The results for $K = 4$, 5, and 6 are shown in Table 4. For a given K and N_{max}, the 50 MHz FPGA decoder outperformed software implemented on the Pentium IV by over two orders of magnitude. The NIOS processor power consumption of approximately 630 mW for all decoders is 30% larger than the highest power consumption for an FPGA decoder (447 mW).

In a hardware experiment we performed a direct comparison between FPGA decode rates on the NIOS board

K	N_{max}	Pentium IV (Kbps)	NIOS (Kbps)	FPGA HW (Kbps)
6	32	0.784	0.003	173.4
6	18	1.344	0.005	228.2
6	12	2.064	0.007	288.6
6	9	2.730	0.009	410.9
6	7	3.382	0.011	447.1
6	6	3.615	0.013	450.2
6	5	4.227	0.015	501.3
6	4	5.294	0.019	469.2
6	3	6.239	0.024	471.4
5	16	1.589	0.006	312.2
5	12	2.048	0.008	301.0
5	9	2.661	0.010	411.3
5	8	3.013	0.012	444.8
5	6	4.160	0.015	468.6
5	5	4.899	0.019	470.1
5	4	5.824	0.022	472.6
4	7	3.177	0.014	487.8
4	6	3.404	0.017	515.6
4	5	4.193	0.020	622.8
4	4	5.241	0.024	688.2
4	3	7.381	0.030	734.6

Table 4. Decoding speed of FPGA ASOVA decoder versus microprocessors

(including PC-to-board transfer overheads) and Pentium IV PC decode rates. When 100 Mbps Ethernet PC-to-board delays are considered, the overall decode speed for a $K = 6$, $N_{max} = 18$ decoder is 211.1 Kbps and the overall decoder speed for a $K = 5$, $N_{max} = 12$ decoder is 229.7 Kbps. These values are still more than two orders of magnitude faster than corresponding Pentium IV PC decode rates of 1.3 Kbps and 2.1 Kbps, respectively.

6.5 Comparison to Commercial Cores

In a final experiment, we compared our ASOVA decoder to comparable commercial FPGA turbo code cores available from Xilinx [26] and Altera [1]. All results in Table 5 are for a (15,13) code with 5 iterations. Xilinx and Altera values were taken from respective data sheets. A power value for the Altera core was unavailable.

7 Related Work

Although some reduced-complexity turbo coding approaches have been explored, published implementation results are scarce. Several path pruning algorithms, similar to ASOVA, evaluate the possibility of eliminating low cost paths. The Soft-output Adaptive Viterbi Algorithm [7] adapts SOVA to include a flexible T value to prune low-cost paths while maintaining BER performance. At each stage, only those paths whose path metrics are higher than the threshold are preserved. As a result, the number of survivor paths is reduced, limiting required computations and memory size. Path pruning approaches based on T and N_{max} have been presented for MAP-based turbo decoders [11]. The *M-BCJR* algorithm preserves a fixed number of the best paths at each trellis stage. The *T-BCJR* algorithm preserves all paths with a path metric above a threshold, T. Results show that the latter approach is more efficient and preserves more paths across iterations. No hardware implementation results for these algorithms have been reported.

A reconfigurable turbo decoder based on MAP was implemented in a ReConfigurable Processor Board (RCP) [14], a PCI board consisting of six Altera FLEX 10K70 FPGAs and SRAM units. The implementation allows for configuration based on constraint length K, interleaver length, and decoding iterations. Run-time dynamic reconfiguration was not used with this system. The reported decoding speed for a $K = 4$ configuration using six iterations is 700 cycles/bit, a decode rate of 71 Kbps at 50 MHz. Another implementation [5] mapped a MAP-based turbo decoder onto a PC-based card. Reported data rates varied between 4.8 Kbps and 128 Kbps.

8 Conclusion

In this paper we have presented a dynamically reconfigurable FPGA-based turbo decoder which has been optimized for power consumption. The component decoding algorithm, ASOVA, has been derived from an existing decode algorithm to provide the reduced complexity necessary for efficient hardware implementation. Parameters for the decoder have been determined via simulation. The key to power savings is decoder reconfiguration based on changing channel noise conditions. Through experimentation it is shown that up to a 52% power savings can be achieved by reconfiguring the decoder at run time rather than requiring a static implementation of a higher-complexity, higher power decoder. Our decoder has been verified in hardware using an Altera NIOS Development Board containing a Stratix FPGA.

In the future, we plan to contrast our decoder to software implementations on low power DSPs that employ voltage scaling. It is possible that noise-based reconfiguration of voltage levels may be possible for both DSP and FPGA-based implementations. We also plan to consider approaches to make the turbo decoder partially reconfigurable.

Model	FPGA	LUTs	MEM (Kbit)	f_{max} (MHz)	Power (mW)	Speed (Mbps)
Xilinx [26]	XC2V500	5390	360	66	970	2.0
Altera [1]	EP1S10	5644	400	50	N/A	2.0
ASOVA	EP1S10	2066	65	76	248	1.4

Table 5. Comparison of ASOVA core and commercial cores for a (15,13) code

9 Acknowledgments

The work was funded in part by National Science Foundation grants CCR-9875482, CCR-9988238, EIA-0080119 and ECS-0300130 and a grant from M/A-COM. The authors wish to thank Altera for the donation of the NIOS Development Board and Quartus II software.

References

[1] Altera Corporation. *MegaCore Function User Guide Turbo Encoder/Decoder*, 2003.

[2] Altera Corporation. *Nios Stratix Development Kit*, 2003.

[3] Altera Corporation. *Stratix Data Sheet*, 2003.

[4] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT-20:284–287, Mar. 1974.

[5] S. A. Barbulescu, W. Farrell, P. Gray, and M. Rice. Bandwidth Efficient Turbo Coding For High Speed Mobile Satellite Communications. In *Proceedings, International Symposium on Turbo Codes and Related Topics*, pages 119–126, Brest, France, Sept. 1997.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes. In *Proceedings, International Conference on Communications*, pages 1064–1070, Geneva, Switzerland, May 1993.

[7] F. Chan. Adaptive Viterbi decoding of turbo codes with short frames. In *Proceedings, Communication Theory Mini-Conference*, pages 47–51, Vancouver, BC, June 1999.

[8] G. C. Clark and J. B. Cain. *Error-Correction Coding for Digital Communications*. Plenum Publishing, New York, NY, 1981.

[9] J. A. Erfanian, S. Pasupathy, and G. Gulak. Reduced complexity symbol detectors with parallel structures for ISI channels. *IEEE Transactions on Communications*, 42(2/3/4):1661–1671, Feb/Mar/Apr 1994.

[10] M. Fossorier, F. Burkert, S. Lin, and J. Hagenauer. On the equivalence between SOVA and Max-Log-MAP decodings. *IEEE Communications Letters*, 2(5):137–139, May 1998.

[11] V. Franz and J. B. Anderson. Concatenated decoding with a reduced-search BCJR algorithm. *IEEE Journal on Selected Areas on Communication*, 16(2):186–195, Feb. 1998.

[12] F. Gilbert, A. Worm, and N. Wehn. Low Power Implementation of a Turbo-Decoder on Programmable Architectures. In *Proceedings, Asia South Pacific Design Automation Conference*, pages 400–403, Yokohama, Japan, Jan. 2001.

[13] J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory*, 42(2):429–445, Mar. 1996.

[14] S. Halter, M. Oberg, P. M. Chau, and P. H. Siegel. Reconfigurable Signal Processor for Channel Coding & Decoding in Low SNR Wireless Communications. In *Proceedings, IEEE Workshop on Signal Processing Systems*, pages 260–274, Cambridge, MA, Oct. 1998.

[15] O. J. Joerssen, M. Vaupel, and H. Meyr. Soft-output Viterbi Decoding: VLSI Implementation Issues. In *Proceedings, IEEE Vehicular Technology Conference*, pages 941–944, Secaucus, NJ, May 1993.

[16] O. Y.-H. Leung, C.-W. Yue, and C.-Y. Tsui. Reducing Power Consumption of Turbo Code Decoder Using Adaptive Iteration with Variable Supply Voltage. In *Proceedings, International Symposium on Low-Power Electronics and Design*, pages 36–41, San Diego, CA, Aug. 1999.

[17] J. Liang. *Development and Verification of a System-on-a-Chip Communication Architecture*. PhD thesis, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, May 2004. Also available as UMass ECE Technical Report TR-04-CSE-04.

[18] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1983.

[19] Micron Technology, Inc. *MT48LC4M32B2 SDRAM Data Sheet*, 2003.

[20] S. Nanda, K. Balachandran, and S. Kumar. Adaptation techniques in wireless packet data services. *IEEE Communications Magazine*, 38(1):54–64, Jan. 2000.

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

[22] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, Upper Saddle River, NJ, 1996.

[23] P. Robertson, E. Villebrun, and P. Hoeher. A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain. In *Proceedings, International Conference on Communications*, pages 1009–1013, Seattle, WA, June 1995.

[24] S. J. Simmons. Breadth-first trellis decoding with adaptive effort. *IEEE Transactions on Communications*, 38(1):3–12, Jan. 1990.

[25] S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson. A Dynamically Reconfigurable Adaptive Viterbi Decoder. In *Proceedings, International Symposium on Field Programmable Gate Arrays*, pages 227–236, Monterey, CA, Feb. 2002.

[26] Xilinx, Inc. *3GPP Turbo Decoder Data Sheet*, 2001.