FPGA Based Storage Efficient Viterbi Decoders

By

Dr. M. O' Droma and J. O'Shea, University of Limerick

Abstract

Field programmable gate arrays (FPGA) provide one of the fastest methods to implement and prototype an Application Specific Integrated Circuit (ASIC). When a hardware description language (HDL) is used to model and synthesize the behaviour of the ASIC, the overall design cycle is reduced. A novel approach to achieving path memory savings was proposed in a previous paper. This paper presents a number of FPGA based traceback Viterbi decoders with this path memory using the Xilinx XCV50 Virtex FPGA device. It is shown that Viterbi decoders using this storage efficient path memory unit require a smaller chip area and achieve a faster decoding time without loss of decoding performance. Decoders using this novel path memory can achieve savings of 20% in storage for (n,1,m) codes, and <=20% for general (n,k,m) codes.

1. Introduction

The Viterbi algorithm is renowned as a maximum likelihood (ML) decoding technique for convolutional codes. The path memory unit in an (n,k,m) Viterbi Decoder is responsible for keeping track of the information bits associated with the surviving paths designated by the path metric unit. In order to track the surviving paths for all states in an ASIC based decoder, a path memory needs to be of length T, where T is a fixed length called the truncation length. A novel approach to achieving path memory savings has been identified, [4], [5] which shows that it is possible to reduce path memory requirements without loss of decoding performance.

An FPGA is a programmable device with an internal array of logic blocks, surrounded on the periphery by programmable input/output blocks. The entire array is connected with programmable interconnect. In this paper the Xilinx XCV50 Virtex FPGA is the target platform. In this platform the basic logic building block is called a configurable logic block (CLB), [16]. Each CLB has two identical slices, and each slice contains two 4-input look up tables (LUTs), two D-type flip-flops with dedicated clock enable, set/reset, and fast carry-in/carry-out signal paths. The 4-input LUTs can perform any 4 variable Boolean function or can be configured as 16x1 RAM.

The decoders were designed using Verilog HDL, synthesized using *Synopsys* FPGA compiler, and the logic placed and routed using the *Xilinx* toolset.

2. Path Memory Architectures

There are two basic design techniques: Register Exchange *(RE)*, and Traceback *(TB)*, [7]. In both techniques a shift register is associated with every trellis node throughout the decoding process. The *RE* approach is quite simple from a functional point of view, but suffers from the disadvantages that every register must go through a Read-Modify-Write (*RMW*) cycle for every branch label written to path memory. In this paper, the branch labels used in the *RE* method are called *forward labels*. In a fast VLSI decoder with a *RE* path memory all of the exchanging must take place simultaneously, leading to hardware intensive design with large power consumption, [7], [12]. Traceback decoders do not require *RMW* operations

but use simple shift register operations, and hence the power consumption for *TB* decoders is considerably reduced due the lower flip-flop toggle rates. This is one of the primary reasons that *TB* decoders are preferred to *RE* Viterbi decoders, as typically these decoders are used in low power applications. The simple shift register operations used in *TB* decoders are a result of a different branch label used. In this paper this branch label is called a *backward label*.

2. Novel Path Memory.

Significant savings in storage can be achieved in Viterbi decoders when using a backward label path memory. This can be implemented by exploiting the trellis property outlined in the (n, k, m), and the (n, 1, m) code theorems, [4], [5].

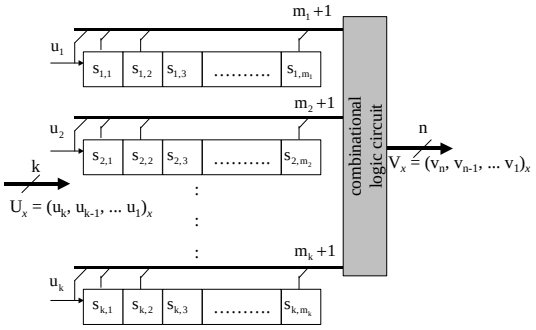

Figure 1 (n,k,m) Convolutional Encoder.

Given the (n, k, m) non-systematic convolutional encoder in figure 1, a stage denotes an instance of the encoding process, i.e. the encoders memory state, its input and its output, which lasts for the duration of one input symbol period. In any stage *x*, the n bits of the output symbol, $V_x = (v_n, v_{n-1}, \ldots v_1)_x$, are given by fixed combinations of selected bits of the input symbol, $U_x = (u_k, u_{k-1}, \ldots u_1)_x$ and selected elements of the encoder memory, i.e. the elements of previous input symbols, [4].

The state of this memory at stage *x* is denoted S_x,
$S_x = (s_{j,i})_x$,where j = k,...1 and i = m_j,...1.

In Viterbi decoder implementations trellis branch labels are used to store information about encoder state in order to decode the surviving paths through the code's trellis. Labels can be forward or backward, the former being identical to the k-tuple input in the same encoder stage, and the latter identical to the k-tuple shifted out of the encoder on transition to the next stage, [4]. Thus at stage *x* forward, (F_x), and backward, (B_x) branch label information may be identified as:

$$\begin{aligned} F_x &= f_{k,x}, f_{k-1,x}, \ldots, f_{1,x} \\ &= (f_k, f_{k-1}, \ldots, f_1)_x \\ &= (u_k, u_{k-1}, \ldots, u_1)_x \\ B_x &= b_{k,x}, b_{k-1,x}, \ldots, b_{1,x} \\ &= (b_k, b_{k-1}, \ldots, b_1)_x \\ &= (s_{k,m_1}, s_{k-1,m_{k-1}}, \ldots, s_{1,m_1})_x \end{aligned}$$

A forward label Viterbi decoder utilizes a path memory that stores k-bit forward labels at every trellis stage. Similarly, a backward label Viterbi decoder utilizes a path memory that stores k-bit backward labels at every trellis stage.

Convolutional codes have the property where every j^{th} element of a forward label branch and a backward label branch in a trellis path are separated by a distance equal to the memory order of the encoder's j^{th} shift register. Any j^{th} element of a forward label branch is equivalent to the corresponding j^{th} element in the backward label branch m_j stages later. Conversely, any j^{th} element of a backward label branch is equivalent to the corresponding j^{th} element in the forward label branch m_j stages earlier. The converse of this

property can be used at any stage *x*, during the traceback process of a backward label decoder. That is, in any stage *x*, a decoder decision can be made on the j^{th} input element of the n-bit symbol, which stretches, back to stage $x+ m_j$. Thus for the j^{th} element of the encoder shift register there is a saving of m_j stages. As the backward label at stage *x* is equivalent to the forward label m_j stages previous, the storage requirements can be reduced for each j^{th} shift register by length m_j without loss of decoding performance.

During traceback, a mapping function is used to trace a state at the current traceback stage S_x to its predecessor state at stage S_{x-1} on the ML path. The mapping function uses the backward label B_x read from path memory to map S_x. to S_{x-1}.
The traceback mapping function is thus defined as:

$$S_{x-1} = f(S_x, B_x),$$

As the number of traceback stages is T, this traceback mapping function is called for T iterations until stage 1 of the path memory register is reached. Once the traceback process has completed, the decoder can release one decoded symbol by selecting the left most elements of the trace back mapping register as the decoded symbol. However when using the novel technique, the same elements are available m_j stages earlier. Depending on the convolutional code used, the decoded symbol elements are either entirely resident in the right most elements of the traceback register, or a fixed combination of the rightmost traceback register elements and selected backward label elements read from memory. Therefore the novel technique requires only a small change to the output decision circuit, i.e. rewiring the required elements to the decoded symbol output.

Effects of "Encoder Flushing" for the Novel Technique

Because the path memory of a Viterbi decoder has fixed length T, the input sequence to the corresponding convolutional encoder is also forced into a block structure by periodic truncation. This is done by appending a number of zeros to the input sequence for the purpose of clearing or flushing the encoder shift register elements, [14]. As the novel technique has m_j less path memory registers than the conventional case, one could ask the question, is "encoder flushing" lost and hence loss of block synchronization between decoding blocks?
Again the savings gained are realized from the backward label elements m_j stages earlier than in the conventional case. Therefore the first m_j stages during traceback are not required to realize the novel technique. This effectively means that in practical implementations, the stages of the novel path memory are synchronized or have a "decoding window" around trellis stages m_j to T. Hence the final m_j stages, which correspond to the "encoder flushing" stages, are preserved and will not lose block synchronization.

3. FPGA Implementation

This section details the hardware design for a number of backward label traceback Viterbi decoders. The hardware decoders are designed to use bit metrics that allow hard decisions to be made on the incoming digital bit sequence. The (2,1,3) and (3,2,2) test codes were used as examples of (n,1,m) and (n,k,m) based decoders cases respectively.

The decoders work on the assumption that all pre-processing activity on the

received sequence has been carried out correctly, and that the decoder receives a fully synchronized digital sequence, i.e. n-bits per trellis stage on every clock cycle. A fully realized Viterbi decoder ASIC would therefore need this preprocessing core, i.e. the front-end receiver, A/D converter and block synchronization functions. The top-level block diagram for the two (2,1,3) decoders, and the associated top-level symbol is shown in figure 2, followed by the top-level signal descriptions in table 1.

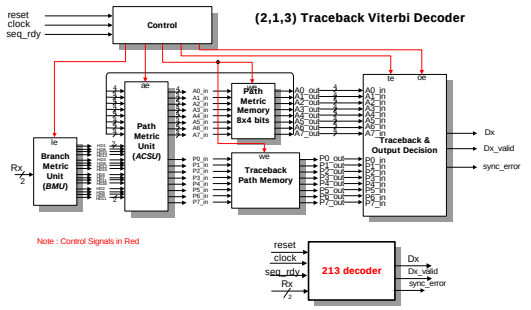

Figure 2 (2,1,3) Viterbi Decoder.

Signal	Direction	Description
Rx[1:0]	input	A 2-bit digital sequence representing the n-bit received sequence for the current trellis stage
seq_rdy	input	A pulse to indicate a new received sequence is ready to be decoded
clock	input	Chip level clock input.
reset	input	Chip level reset input.
Dx	output	The decoded symbol output, which can be either a zero or a one and qualified by the assertion of *Dx_Valid*. If *Dx_Valid* is de-asserted this signal remains in tri-state.
Dx_Valid	output	An output signal used to qualify a valid decoded symbol is available
sync_error	output	An output signal to indicate that the decoder is out of sync with the incoming received sequence

Table-1 Top-level Signals (2,1,3) Decoder.

For the (2,1,3) decoder, the conventional path memory is organized as eight parallel shift registers (P0-P7), each of length fifteen, i.e. (2^M x 5m). At every trellis stage eight backward labels are produced by the ACSU, if write enable (*we*) is asserted these backward labels are shifted into the corresponding path memory shift register with the existing register contents moving one place to the right. A path memory write-pointer is incremented each time a set of backward labels is shifted in.

Once the path memory is full, the traceback process can begin. Figure 3 illustrates the functional blocks in the (2,1,3) conventional traceback and output decision block.

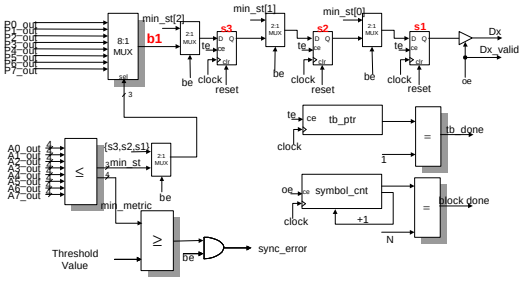

Figure 3 (2,1,3) Output Decision Unit.

The operation is as follows. First, the minimum path metric state is determined by taking the eight path metrics as inputs to a ≤ comparator, and the comparator outputs the corresponding state number, *min_st*. If the best state enable (*be*) signal is asserted, the traceback mapping register is then loaded with this state number, *min_st*. This state number is also used as the select bits for an 8:1 MUX. This MUX takes all eight backward labels from path memory, and selects the backward label corresponding to the state number. Finally the traceback pointer (*tb_ptr*) is loaded with a value that points to the last entry in the path memory. Traceback then proceeds once the traceback enable signal (*te*) is asserted.

The storage efficient decoder only requires a path memory with three less storage stages, for a total saving of M.2^M = 24 memory elements. Figure 4 illustrates the functional blocks in the traceback and

output decision block for the (2,1,3) efficient decoder. The operation of tracing back is identical to the conventional case. However, the traceback operations terminate earlier than the conventional case because of the smaller memory requirements. In addition, the output decision circuit is wired differently than the conventional case in order to take advantage of the efficient traceback architecture. In the efficient implementation, the output decision is wired to the *s3* register instead of the *s1* register in the conventional case, as shown in figure 4.

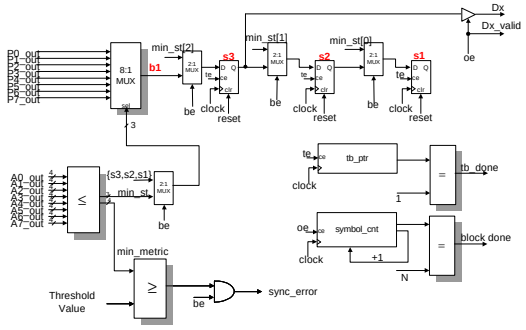

Figure 4 Efficient (2,1,3) Output Decision Unit.

(3,2,2) Decoders

The second two designs discuss the implementation details for the (3,2,2) conventional and efficient decoders. As in the (2,1,3) case, the decoders work on the assumption that all pre-processing activity on the received sequence has been carried out correctly, and that the decoder receives a fully synchronized digital sequence. The top-level block diagram for the two (3,2,2) decoders is shown in figure 5, followed by the top level signal descriptions in table 2.

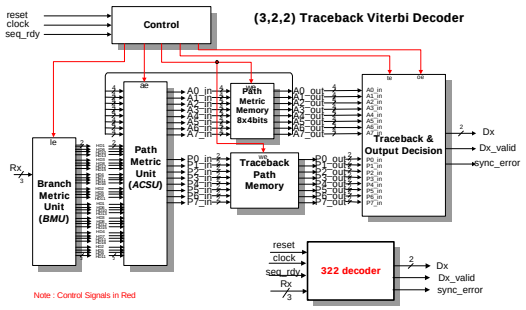

Figure 5 (3,2,2) Viterbi Decoder.

Signal	Direction	Description
Rx[2:0]	input	A 3-bit digital sequence representing the n-bit received sequence for the current trellis stage
seq_rdy	input	A pulse to indicate a new received sequence is ready to be decoded
clock	input	Chip level clock input.
reset	input	Chip level reset input.
Dx [1:0]	output	The 2-bit decoded symbol output. A valid decoded symbol is qualified by the assertion of the *Dx_Valid* signal. Otherwise the signal remains in the tri-state condition.
Dx_Valid	output	An output signal used to qualify a valid decoded symbol is available
sync_error	output	An output signal to indicate that the decoder is out of sync with the incoming received sequence

Table-2 Top-level Signals (3,2,2) Decoder.

The (3,2,2) conventional path memory is organized as sixteen parallel shift registers (2 per state), where each register is of length ten, i.e. (2^M x 5m). At every trellis stage eight 2-bit backward labels are produced by the ACSU, if the write enable (*we*) is asserted these backward labels are shifted into the corresponding path memory shift registers with the existing register contents moving one place to the right. A path memory write-pointer is incremented each left shift (write) operation. The (3,2,2) traceback and output decision block is shown in figure 6.

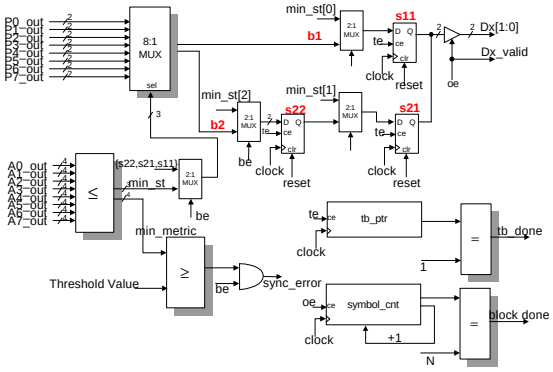

Figure 6 (3,2,2) Output Decision Unit.

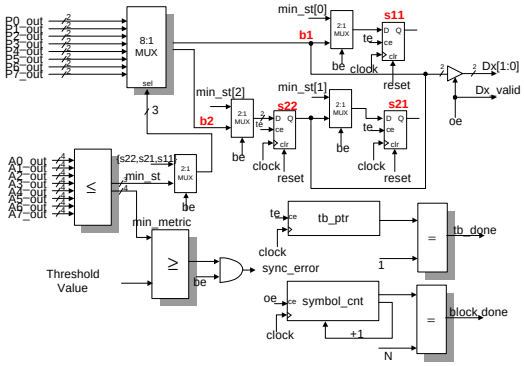

Figure 7 Efficient (3,2,2) Output Decision Unit.

The operation of selecting the minimum path metric state and loading the traceback register is identical to the (2,1,3) case. For every clock cycle in the traceback operation, the backward label bits {*b2*,*b1*} are read from path memory and shifted into the traceback register elements *s22*, and *s11* respectively. Once the traceback pointer terminates, the control block asserts the output enable (*oe*) signal to output a decoded symbol from the *s21*, and *s11* elements of the traceback mapping register.

For the (3,2,2) efficient decoder, each state path memory register pair is of length eight and nine. The operation of shifting in backward labels is identical to the conventional case. Again the output decision block for the efficient (3,2,2) decoder has the same functional blocks as the conventional case. In the efficient implementation, the most significant bit is wired to *s22* instead of *s21*, and the least significant bit is wired to *b1* instead of the *s11* in the conventional case. The traceback and output decision block is shown in figure 7.

Implementation Results

Each CLB in a Xilinx XCV50 FPGA has two identical slices. Each slice comprises of two 4-input look up tables (LUTs), and two D-type flip-flops. The actual synthesized circuits for both the conventional and efficient designs will thus be reported in terms of the number of LUTS and DFFs used per module. As the decoder designs are modularized, some of the building blocks are identical between both implementations. Table 3 and 4 provides the logic utilization comparison between both implementations.

Module	Conventional Design		Efficient Design	
	LUTs	DFFs	LUTs	DFFs
BMU[1]	9	32	9	32
ACSU[2]	184	64	184	64
Path Memory	0	120	0	96
Path Metric Memory	0	32	0	32
state_sel[3]	99	19	99	19
CTDU[4]	200	34	157	34
TOTAL	483	301	449	277

Table 3 (2,1,3) Utilization.

[1] BMU - Branch Metric Unit
[2] ACSU – Add Compare Select Unit
[3] state_sel – Minimum Path Metric State Selector
[4] CTDU – Control, Traceback, and O/P Decision unit

Module	Conventional Design		Efficient Design	
	LUTs	DFFs	LUTs	DFFs
BMU	16	62	16	62
ACSU	448	128	448	128
Path Memory	0	160	0	136
Path Metric Memory	0	32	0	32
state_sel	99	19	99	19
CTDU	207	34	216	38
TOTAL	770	435	779	415

Table 4 (3,2,2) Utilization.

Note: The Verilog designs are not design dependent on the Xilinx FPGA platform. Other FPGA vendors such Altera or Lattice could also have been used; however access to these design tools was not available.

4. Design Flow

For this project, Verilog HDL was used to design the Viterbi decoders. The design flow using Verilog is typical of many FPGA design flows and is shown below in figure 8.

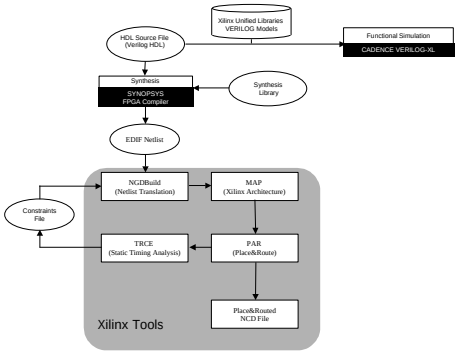

Figure 8 FPGA Design Flow.

The Cadence Verilog-XL tool was used to check the Verilog code syntax, and to perform functional verification of the designs. Typically the verification step is carried out with a Verilog based test-bench program. For this project a test bench was written to stimulate the decoders under test with known convolutional coded test sequences. The decoded sequences were then verified against known good convolutional decoded sequences.

The designs were synthesized using *Synopsys* FPGA compiler, which takes the simulated Verilog code and generates an EDIF net list. The Xilinx backend tools are then used to translate the EDIF file into a functional FPGA design.

5. Decoding Performance

This section discusses some metrics for evaluating the decoding performance of the Viterbi decoders in this paper. It is demonstrated that Viterbi decoder implementations using the novel path memory have a faster decoding time.

Once a received sequence is ready to be decoded, the decoder will perform the following operations for each trellis stage until the path memory is full:

- 1 clock cycle to compute branch metrics.
- 1 clock cycle to compute path metrics, and backward labels.
- 1 clock cycle to store the path metrics, and backward labels.

These three operations are repeated for each of the T stages until the path memory is full. Once the path memory is full, it takes 1 clock cycle to register the minimum path metric state, followed by T clock cycles for traceback. Once the traceback pointer reaches the beginning of path memory the first decoded symbol is then determined from the traceback mapping register. Therefore the number of clock cycles consumed before the first

symbol is released by the decoder amounts to 3T + (T+1) clock cycles.

After the first decoded symbol is produced, the oldest path memory contents are shifted out as the next set of backward labels are shifted in. The corresponding path metric contents are also updated. The path memory is now full again. The state with the minimum path metric is registered and traceback recommences. Therefore the number of clock cycles consumed for each of the remaining N-1 symbols is (T+4) clock cycles per symbol:

- 1 clock cycle to compute the branch metrics
- 1 clock cycle to compute the path metrics, and backward labels
- 1 clock cycle to store the path metrics, and backward labels
- 1 clock cycle to register the minimum path metric state
- T clock cycles to perform traceback

An expression for the total number of clock cycles consumed to decode an N-bit sequence is now derived. Given the truncated path memory length T, the total number of clock cycles, *C* consumed by a conventional Viterbi decoder to decode an N-bit sequence is:

$$C = 3\mathrm{T} + (\mathrm{T}+1) + \left(\prod_{\mathrm{i}=1}^{\mathrm{N-1}} (\mathrm{T}+4) \right)$$

Example 1:
Given the conventional (2,1,3) decoder, the total number of clock cycles consumed to decode an entire sequence of 225 symbols is computed as follows.
There are fifteen path memory stages; therefore the total number of clock cycles taken to fill path memory is 3T or 45. Once the path memory is full, the decoder takes one clock cycle to register the minimum metric state, and finally fifteen clock cycles to traceback through the path memory, a decoded symbol is then released. This is repeated for every symbol decoded, and the total number of clock cycles amounts to:

$$C_{b213} = 3\mathrm{T} + (\mathrm{T}+1) + \left(\prod_{\mathrm{i}=1}^{\mathrm{N-1}} (\mathrm{T}+4) \right)$$
$$C_{b213} = (61) + \left(224(19)\right)$$
$$C_{b213} = 4{,}317$$

For the (2,1,3) efficient backward label decoder, there are twelve path memory stages. The total number of clock cycles *C* consumed by an efficient Viterbi decoder to decode the same sequence is:

$$C_{e213} = 3\mathrm{T} + (\mathrm{T}+1) + \left(\prod_{\mathrm{i}=1}^{\mathrm{N-1}} (\mathrm{T}+4) \right)$$
$$C_{e213} = (49) + \left(224(16)\right)$$
$$C_{e213} = 3633$$

This is a percentage gain in decoding performance of approximately 16% for the (2,1,3) efficient decoder.

Example 2:
Given a conventional (3,2,2) decoder, the total number of clock cycles consumed to decode an entire sequence of 150 symbols is computed following the same process as the (2,1,3) example.
Therefore the total number of clock cycles amounts to:

$$C_{b322} = 3\mathrm{T} + (\mathrm{T}+1) + \left(\prod_{\mathrm{i}=1}^{\mathrm{N-1}} (\mathrm{T}+4) \right)$$
$$C_{b322} = (41) + \left(149(14)\right)$$
$$C_{b322} = 2{,}127$$

For the (3,2,2) efficient backward label decoder, there are a maximum of nine stages of path memory, and the total

number of clock cycles consumed to decode the same sequence is:

$$C_{e322} = 3\mathrm{T} + (\mathrm{T}+1) + \left(\prod_{\mathrm{i}=1}^{\mathrm{N-1}} (\mathrm{T}+4) \right)$$
$$C_{e322} = (37) + (149(13))$$
$$C_{e322} = 1{,}974$$

This is a percentage gain in decoding performance of approximately 7%.

6. Conclusions

In this paper a storage efficient hardware architecture for Viterbi Decoders using hard decisions has been presented. This novel implementation presents 20% savings in storage requirements for all (n,1,m) codes, <=20% for (n,k,m) decoders. The (3,2,2) decoder produced 15% savings in path memory storage requirements. The particular decoder design used hard decisions, however identical gains are also achievable for soft decision decoding. For 2^b level quantized soft decision decoding, *b* bits are used to describe each symbol. Therefore the path memory for soft decision decoder requires *b* times the capacity of the equivalent hard decision decoder. However, this increase in path memory for soft decision decoding increases per symbol only and does not increase the decoding depth for ML decoding. Therefore the path memory savings in decoding depth will benefit hard decision and soft decision based Viterbi decoders.

The novel decoders also provide a faster decoding time than the conventional case. A metric for measuring decoding performance is presented, and it is shown that the decoding performance for the novel technique is improved significantly. The novel (2,1,3) decoder has a performance gain of 16%, with the novel (3,2,2) decoder a performance gain of 7%.

This novel path memory technique can be easily incorporated into existing Viterbi decoder architectures with minimal effort for significant storage and performance gains.

References.

[1] A.J. Viterbi, *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm.* IEEE Transactions on Information Theory, Vol. IT-13, April 1967, pp. 260-269.

[2] G.D. Forney Jr. *The Viterbi Algorithm.* Proc. IEEE Vol. 61, 1973, pp. 268-278.

[3] J.K. Omura, *On the Viterbi Decoding Algorithm.* IEEE Transactions on Information Theory, Vol. IT-15, 1969, pp. 177-179.

[4] M. O'Droma, et al, *More Efficient ASIC Implementation of digital radio CODECS.* I.E.E.E. 1997

[5] M. O'Droma, et al, *Memory Savings in Viterbi decoders for (n,1,m) convolutional codes*. I.E.E 1997.

[6] M. O'Droma, et al, *VLSI Design of a of GSM channel decoder*. Report, University of Limerick and Teltec Irl., Ireland, July 1993.

[7] S. Wicker, *Error Control Systems for Digital Communication and Storage.* Prentice-Hall Inc., 1995

[8] T. Truong, et al, *A VLSI Design for a Trace-Back Viterbi Decoder* IEEE Transactions on Communications, Vol. 40, No.3, March 1992, pp. 617-624.

[9] R. Cypher, et al, *Generalised Trace-Back Techniques for Survivor Memory Management in the Viterbi Algorithm.* Journal of VLSI signal processing, 5, Jan 1993, pp85-94.

[10] M. Horwitz, et al, *A Generalised Design Technique for Traceback Survivor Memory Management in the Viterbi Decoders*. Proc. I.E.E.E, 1997

[11] S. Jung, et al, *A New Survivor Memory Management Method in Viterbi Decoders*. Proc. I.E.E.E, Vol. 1 1996, pp.126-130.

[12] C. Rader. *Memory Management in a Viterbi Decoder*. IEEE Transactions on Communications, Vol. 29, no.9, Sept 1981, pp. 1399-1401.

[13] G. Feygin, et al, *Survivor Sequence Memory Management in a Viterbi Decoder*. Proc. IEEE, Vol. 5, June 1991, pp. 2967-2970.

[14] B. Sklar, *Digital Communications: Fundamentals and Applications.* Prentice-Hall Inc., Englewood Cliffs, N.J., 1988

[15] Lin and Costello, *Error Control Coding: Fundamentals and Applications.* Prentice-Hall Inc., Englewood Cliffs, N.J., 1983

[16] Xilinx, *The Programmable Logic Data Book*, 2000

[17] Altera, *The Programmable Logic Data Book*, 2000