Low Complexity FPGA Implementation of Register Exchange Based Viterbi Decoder

Abdulrazaq Muhammad B., Abdullahi Zanna M., Almustapha Mohammed D, Dajab Danjuma D.
Department of Electrical and Computer Engineering,
Ahmadu Bello University Zaria, Nigeria.
Email: mbashiray@yahoo.com, zabdullahi@abu.edu.ng, md.almustapha@gmail.com, dddajab@abu.edu.ng

***Abstract*- Viterbi decoders are either implemented as a Trace Back model or Register Exchange model. The trace back model is more popular, despite having large latency, because it requires less power and has less hardware complexity. In this paper a low complexity FPGA implementation of Register Exchange method is presented which gave the same error performance as that of trace back method at the same time having less hardware requirement. This is achieved using continuous decoding that normalises the state metric for every symbol processed, which reduces the size of registers used for storing the state metric. Split search that finds the best path every clock cycle was used for the normalisation and choosing the data output. This makes it possible to have output rate of one (1) symbol which reduces the number of registers required to save the path history bits to be outputted and reduces delay. Result of implementation of rate half codes of constraint length 7 and generator vector [1111001, 1011011] shows that for the same performance, the Register Exchange implementation presented, used no RAM as in the case of normal trace back and one-pointer algorithm. It requires less silicon area (5.5% less FPGA slices). It can run as fast as or even faster than the latter implementations and has less latency (1-decoding depth as compared to 4-decoding depth of trace back method) than the rest.**

***Keywords– Forward Error Correction (FEC), Convolution, Viterbi Algorithm (VA), Trellis, Constraint Length, State Exchange, Register Exchange, Trace back*.**

I. INTRODUCTION

Digital signal sent from one communication channel to another undergo some undesirable changes which brings errors in the data at the receiving end. In order to combat the errors, different techniques are used such as block coding and convolutional coding. These are called Forward Error Correction (FEC) codes. Prior to sending the data, the coding is carried out, to be able to identify errors at the receiver. This technique employs the addition of some patterned redundant bits to the message bit so that if an error occurs, it can be detected and possibly corrected.

Convolutional coding is one of the frequently used FEC [1]. Viterbi decoding is an optimal decoding algorithm that is used to decode convolutional codes. It takes a certain number of bits called the trace-back length (D), compare it with all the known allowable sequences that may come out of the convolutional encoder, compute its distance from the sequences and choose the sequence that is closest to the received data.

Viterbi decoding suffers high delay before output is produced when trace-back (TB) method is used or high power consumption and complexity when Register Exchange (RE) is used.

Several attempts were made to reduce the delay without greatly increasing the power consumption and complexity. The most common approach is modification of TB method [2] [3] [4] [5] [6] and [7]. This paper takes the Register Exchange method and implemented it on FPGA, and showed that it can be made less complex than conventional TB method.

In Section II, an overview of convolutional code and Viterbi decoding is given with a short review of related works that try to reduce the delay of the decoder. In Section III the proposed algorithm is presented. Section IV presents the evaluation of the solution presented. Section V presents the result of the simulated implementation, while Section VI gives the final conclusion of the paper.

II. OVERVIEW OF CONVOLUTIONAL CODING AND VITERBI DECODING

A. Convolutional Encoder

Convolutional encoder is a system of coding in which the output of an encoder depends on the current input and a certain number of previous inputs. It employs the use of modulo 2 adders (XOR gates) to combine selected previous output. The bits added, are selected according to generator matrix (G) which specifies the position of the bits to be selected for each output. G = [101, 111] indicates that to produce two outputs (separated by comma) the first output should be a modulo 2 addition of the first and the third bit in the sequence (101), and the second bit is the addition of all the three bits in the sequence (111). A 1 means the bit will be used and 0 means the bit will not be used. The G is usually written in octal numbers rather than binary. Therefore the above example can be written as G = [5, 7]. Figure 1 shows an example of rate R= 1/2 encoder with constraint length k=7

and a generator matrix G = [171, 133] = [1111001, 1011011], the one used in this paper.

The number of outputs for any given input determines the rate of the code. If for example 2 outputs are generated for any 1 input, the code rate is ½. For a constraint length of k, the number of memory units required is m=k-1 to hold the previous bits for use.

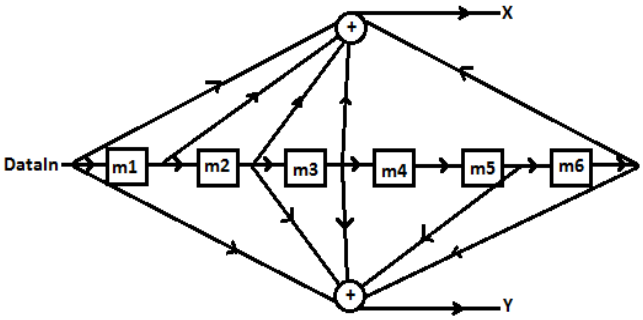

Figure 1: Convolutional Encoder.

B. Viterbi Decoding

Viterbi decoding is one of the most commonly used decoding techniques for decoding convolutional codes. But its complexity exponentially increases with increase in the constraint length of the encoder. This limits its usage to moderate length such as k=9[1].

The decoding uses the fact that there are 2^m number of binary states in which the decoder can be, depending on the bits in the memory units of the encoder. From which ever state it is, the code can move into one of two states when either a 0 or a 1 is at the input. For each of the next state, therefore, two possible states can transit into it. Viterbi decoder selects one of the two states that have the lowest cumulative cost (state metric).

Viterbi decoders are functionally divided into three units. Branch Metric Unit (BMU), Add Compare Select Unit (ACSU) and the Trace-Back Unit (TBU). Fig 2 illustrate the functional set up.

The BMU compares the received signal to all possible input signals. For rate half, where two bits represent a symbol, the possible inputs to the BMU are 00, 01, 10, and 11. The input may be hard measurements of either 0s or 1s, or soft measurements which reflect how close it is to the 0 or 1. The metric is either the hamming distance of the received data from the possible inputs or the minimum mean square distance when soft decision is being used. These metrics are passed to the ACSU.

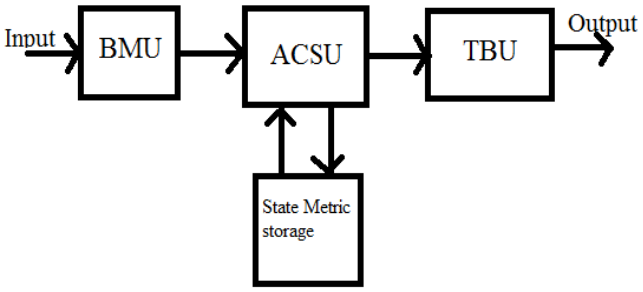

Figure 2: Functional Block Diagram of Viterbi Decoder.

The ACSU adds the appropriate branch metric to the cumulative state metric for the any possible input. The metrics of the two possible states that transit to a given next state are compared and the most probable is selected based on the state metric. The information of which state is selected is kept as survivor path metric (history) of the next state.

TBU is used to parse through the history from a known state or a state determined as having the most probable state metric, and then working backward to the beginning of the sequence to identify the surviving path, through the history of length D (D=trace back length) kept. The selected code sequence is then decoded and the result presented as output.

The TBU is very useful in reducing the power consumption of the decoder. This is because the task of finding the best state out of the 2^m states has to be done only once for every block of data in continuous decoding. It is not even needed for blocked decoding with known finish state. But the processes involved may cause delay of up to 4 time trace back length (4D).

The search for best state is normally part of the code that increase the area used by Viterbi decoders. This is because there will be need for at least 2^m-1 comparisons. If done in parallel, huge area is used. If done in series, large amount of extra latency is introduced. This is the rational for using block of codes with predetermined finish state, so that the point to start searching is known. But block codes needs large registers for storing the state metric.

Register exchange method offers a solution to the delay caused due to trace-back by carrying the information of any state along with the state metric as it transits from one state to another. It suffers larger complexity because of the need to have multiplexers that will be switching the registers' contents for each state. The wiring connecting the registers, multiplexers and other components is also more complex. This large amount of switching and the operations of shift registers bring about high energy consumption. This is why

more emphasis is given to modifying TB method by researchers.

In [2] modifications of TB called 1-pointer algorithm, k-pointer even algorithm and k-pointer odd algorithm were presented and compared. They harness the fact that, there are larger numbers of write operations to memory than read operation, needed within the same time, and therefore share the time accordingly. The 1-pointer is shown to be better in terms of low latency and less memory requirement than the others. In [3] the 1-pointer algorithm was implemented on FPGA, which shows how it has reduced latency to 1.5D with moderate increase in complexity compared to conventional trace-back method and less power consumption.

Reference [4] proposed a low latency Modified State Exchange (MSE) which uses pre-trace back and 3-pointers to achieved a latency of 1.5D. When implemented on CMOS 0.18µm Technology, it showed less complexity and less energy consumption than conventional TB method. It was not still able to achieve the low latency of RE.

Reference [5] used k-layered (k=constraint length) look ahead method to reduce ACSU pre-computation latency. But this is achieved at the expense of increase in complexity.

But [6] improved on the work of [5] which reduces the complexity and in some cases reduces the latency by optimising the number of look-ahead.

Another technique used to reduce complexity includes Adaptive Viterbi Algorithm (AVA) [7] but does nothing to reduce the latency of TB.

However [8] reduced the power consumption of the RE based decoder by using pointers to the registers, rather than the content, in moving the information in the register around. He achieved 23% power reduction compared to VD implementation of 3G described in the literature.

III. Proposed Register Exchange Implementation

For the rate half code there are two bits received for each symbol. This means there are two clock cycles in which to process the data before the next set of bits are received.

At the instance the second bit of a symbol is received (first clock cycle of the process), the two bits that make the symbol are used to calculate the four branch metrics (BMU process).

In the next clock cycle, these branch metrics are used to calculate $2*2^m$state metrics; half of which represent zeros from the 2^m states and the remaining four representing ones from the same 2^m states. Within the same clock cycle, each two State Metrics that go to the same Next state are compared and the best is selected. If a state from the upper2^m metrics is selected, a zero is saved in the register that is associated to the Present State that has the particular State Metric else a one is saved in the register. At the same time the content of the selected register is moved to the next state to which it transits.

In the next clock cycle, in which the next Branch Metric is being determined, the state Metrics of the 2^m surviving states are compared to find the minimum, which is used to normalise the state metrics in future. This prevents the use of large registers for storing the ever increasing metrics. The minimum metric also points the best state whose data will be sent to the output.

The search for the minimum is split into 8 groups for the 64 state metrics implemented here in. The minimum of each 8 is found concurrently in one clock cycle. In the next clock cycle the minimum of the resulting 8 minima is determined and at the same time used to determine data output. Table I shows a depiction of how time is shared for the different operations.

TABLE I Time Sharing.

	Present symbol	Previous symbol
1st bit		
2nd bit	Find BM of symbol 1	
3rd bit		Find State metrics (Symbol 1) Find surviving states Update history
4th bit	Find BM of symbol 2	Find Minimum metric (Symbol 1) Determine data output.
5th bit		Find State metrics (Symbol 2) Find surviving states Update history
6th bit	Find BM of symbol 3	Find Minimum metric(Symbol 2) Determine data output.

To explain how the algorithm operates, for a decoder of convolutional code of constraint length k=3 (m=2) there will be 4 (2^m) states. Let the 4 states be named S00, S01, S10 and S11. This means 4 registers will be used for storing the path history. Let us name the registers R00, R01, R10 and R11. Figure 3 shows the trellis diagram of the state exchange and the register alongside the trellis diagram.

When Branch Metrics are determined, they are added accordingly to Present State Metrics S00 to S11 for the 8 possible paths. The path metric of S00 transiting to Next State S00 is compared to Present State Metric of S10 transiting to Next State S00. If say, the smaller one is the metric of Present State S10, then that metric is stored as the metric of Next State S00 and the content of register R10

moved to register R00. Also the register is shifted and a 1 is added to it. If conversely, it was the metric of Present State S00 that is the minimum, its metric is made the metric of Next State S00 and a 0 is saved in the history.

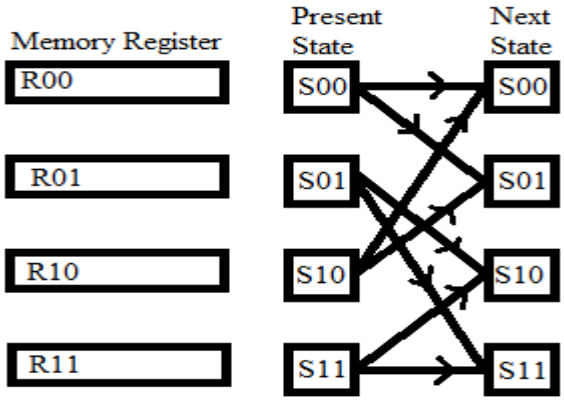

Figure 3: Trellis diagram.

A pseudo code representing the processes can be depicted as below.BM means branch metric, S## represent state metric, R## represent Register, and ## represent any register or state number.

```
1st clock cycle{
        Calculate new BM00, BM01, BM10and B11;
        If (S## is Minimum) then
        Data Output= MSB of R##;
        end if}
2nd clock cycle{
        if(BM00+S00<=BM11+S10)then
        Update history(R) with a 0;
        S00=BM00+S00-MinimumMetric;
        else
        Update history with(R) a 1;
        S00=BM11+S10-MinimumMetric;
        end if
        if(BM11+S00<=BM00+S10)then
        Update history with a 0;
        S01=BM11+S00-MinimumMetric;
        else
        Update history with a 1;
        S01=BM00+S10-MinimumMetric;
        endif}
```

The 2nd clock cycle shows one butterfly operation for determining the next state S00 and S01 from Present state S00 and S10.For this example, there will be two such butterfly modules.

The above algorithm was implemented on Spartan 3E Xilinx FPGA. The code used for encoding and decoding is of constraint length k=7 (m=6), rate ½ and generator matrix G= [171,133] [9].

Soft decision of 3 bit quantisation width was used. In determining the Branch Metrics, Two' Compliment of the received signal (range 000 to 111) with 000 (strongest zero) and 111 (strongest 1) is taken for the four possible combinations of input (00, 01, 10, 11).

Owing to the nature of the operation of FPGA, when a change on a signal is done, it takes effect the next clock cycle. This means even though the register is change in the second clock cycle, the change takes effect in the next first clock cycle. Therefore minimum path metric determination was split into two. It is grouped into groups of eight's in the first clock cycle. This enables the comparisons to be done in parallel. Eight minimums are returned which are compared to find the ultimate minimum in the next cycle during which it is used to access the right output data. It is also used in determining the next state metric by subtracting the minimum to prevent register overflow.

Each butterfly module is run as a concurrent process to speed up the process. This means 32 separate butterfly modules had to be written in the code.

IV. Evaluation of the Design

The design has advantage of low complexity, compared to normal implementation of Register Exchange due to the following features.

Finding the best path, which involves comparing all the state metrics, is one of the processes that increase hardware usage, reduce speed and increase power consumption. This is avoided in a lot of implementations of trace-back based Viterbi decoders by finding the best path only after a group of bits are processed (number of bits at a time is the output rate). To reduce the hardware requirement the 64 states are grouped into 8 groups. The best path of each group is found simultaneously then compared to each other in the next clock cycle, which reduces delay and increases speed. The grouping also significantly reduced the hardware requirement.

The manageability of the process of finding the best path makes it possible to find the best path (minimum metric) for every symbol (2 bit of code) received. This gave the possibility of subtracting the minimum metric (normalising) in every metric update to prevent overflow of metric registers or use of large registers. This also helped reduce the complexity.

Determination of best path for every symbol received gave possibility of having output rate of 1 bit. This reduced the registers required to store the output bits, at the same time reduced the possibility of an error in one output bit to affect other output bits in the same group, thereby needing shorter decoding depth to have similar performance with an equivalent trace back based decoder.

V. RESULTS

The simulation of the VHDL code was done on Xilinx ISE 10.1 with Spartan 3E as the target device. Table II shows the hardware resource utilisation after the simulation of rate ½ code of generator vector [171, 133]. The RE decoder reached optimal performance with decoding depth of 3k (k=constraint length) and 3 bit soft decision. It shows 75% total utilisation of the Spartan 3E slices.

TABLE II HARDWARE REQUIREMENT

Logic Utilisation	***Used***	***Available***	***Utilisation***
No of Slices	3524	4656	75%
No of flip flops	2356	9312	25%
No. of 4 input LUTs	6261	9356	67%
No. of bonded IOBs	8	232	3%
No. of GCLKs	1	24	4%
Maximum Frequency	117.9MHz	300MHz	39%

Where LUTs is Look-Up Tables, IOBs Input-Output Blocks and GCLKs is Global Clocks.

A comparison of the resource usage of the implementation of this algorithm to one pointer algorithm [3] and conventional trace back [10] is shown in Table III.

TABLE III COMPARISON TO 1-POINTER AND CONVENTIONAL TB

Logic Utilisation	**Spartan 3E (This design)**	**Spartan 3E (One Pointer)**	**Spartan 6 (TB design)**
Block RAM	0	2	2
No. Of Slices	3524	3731	Not Available
No. of flip flops	2356	Not Available	4581
Maximum Frequency	117.9MHz	97.0MHz	126.0MHz
Latency	D + k	1.5*D + Output Rate	4*D + k + OutputRate

The comparison shows that the Register Exchange implementation requires even less resources and can run almost as fast as conventional Trace Back design that was implemented on a faster Spartan 6 FPGA [10]. It runs faster than One-Pointer implementation of [3] on the same Spartan 3E FPGA has less latency and less hardware requirement.

VI. CONCLUSION

In this paper, a low complexity FPGA implementation of RE Viterbi decoder is presented. Despite the search for minimum state metric for every symbol received, despite the multiplexers and extra wiring needed for register exchange, a less complex circuit was achieved in the implementation presented. Split search was used, which greatly reduce the complexity of the decoder. Continuous normalisation and output rate of one increased performance per complexity. The result of comparison to conventional trace-back method and 1-pointer algorithm showed that the register exchange implementation presented is about 5.5% less complex and still runs at similar speed with up to 4 times less latency. In future work, this low complexity technique will be used on Modified Register Exchange to face the constraint of power consumption.

REFERENCES

[1] Core Technologies. "Viterbi Algorithm for Decoding Convolutional Codes." Available at: http://www.1-core.com/library/comm/viterbi/ . 2009.

[2] G. Feygin and P. G. Gulak, "Survivor Sequence Memory Managementin Viterbi Decoders," *CSRI Tech.Rep.262, Univ.of Toronto,* Jan-1991.

[3] N. Bhatt., M. Shah, B. Asodariya, "FPGA Implementation of Power Efficient Low Latency Viterbi Decoder". Int. Journal of Eng. Res.& Tech. (IJERT) Vol. 2 Issue 5, May-2013.

[4] Y. Tang, D. Hu1, W. Wei1,W. Lin and H. Lin "A Memory-efficient Architecture for Low Latency ViterbiDecoders" IEEE transaction, June-2009.

[5] J. J. Kon, and K. K. Parhi, "Low-Latency Architectures for High-Throughput Rate Viterbi Decoders", IEEE Ttrans. on VLSISystems, Vol. 12, No. 6, June-2004.

[6] C. Chengand and K. K. Parhi "Hardware Efficient Low-Latency Architecture for High Throughput Rate Viterbi Decoders" IEEE Transon Circuits & Systems-II: Express Briefs, Vol. 55, No. 12, Dec-2008.

[7] S. Swaminathan, R. Tessier, D. Goeckel and W. Burleson, "A Dynamically Reconfigurable Adaptive Viterbi Decoder" FPGA'02Monterey, California, USA, Feb-2002, ACM 1-58113-452-5/02/0002.

[8] D.A. El-Dib and M. I. Elmasry, "Modified Register-Exchange Viterbi Decoder for Low Power Wireless Communications". IEEE Transactions on Circuits and Systems I: Vol. 51, Issue 2. 2004

[9] IEEE. "IEEE standard for Local and Metropolitan Area Networks part 16: Air Interface and Broadband Wireless Access" (802.16-2009 section 8.3.3), 2009

[10] Xilinx. "LogiCORE IP Viterbi Decoder v7.0". Available at: http://www.xilinx.com/support/documentation/ip_documentation/viterbi_ds247.pdf, 2011