IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 4, Issue 5, Ver. I (Sep-Oct. 2014), PP 46-53
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

FPGA Implementation of Viterbi Algorithm for Decoding of Convolution Codes

K.Santhosh Kumar[1], M.V.H. Bhaskara Murthy[2]
[1]*(Department of E.C.E, Aditya Institute of Technology And Management, India)*
[2]*(Department of E.C.E, Aditya Institute of Technology And Management, India)*

Abstract: *Convolutional code is a coding scheme used in communication systems including deep space communications and wireless communications. It provides an alternative approach to block codes for transmission over a noisy channel. The block codes can be applied only for the block of data. The Convolutional coding has an advantage over the block codes in that it can be applied to a continuous data stream as well as to blocks of data. Viterbi decoder employed in digital wireless communication plays a rife role in the overall power consumption of trellis coded modulation decoder. Power reduction in Viterbi decoder could be achieved by reducing the number of states. A pre-computation architecture with T-algorithm was implemented for this purpose, and when we compare this result with full Trellis Viterbi decoder, this approach significantly reduces power consumption without degrading decoding speed. Convolutional encoding with viterbi decoding is a powerful FEC technique that is particularly suited to a channel in which the transmitted signal is corrupted mainly by Additive White Gaussian Noise (AWGN). It operates on data stream and has memory that uses previous bits to encode. The Viterbi Algorithm (VA) is proposed, used for decoding a bit stream that has been encoded using FEC code. The Convolutional encoder adds redundancy to a continuous stream of input data by using a linear shift register. A pre-computation architecture with Viterbi algorithm is implemented for this purpose, Viterbi (Convolutional) encoder and Viterbi decoder are designed and implemented using FPGA technology, which are the essential blocks in digital communication systems. It is particularly suited to a channel in which the transmitted signal is corrupted mainly by AWGN. The Viterbi decoder of Constraint length 7 and code rate ½ is considered. The design is implemented using verilog on Xilinx Spartan 3E and advanced Spartan 6 board and the results and Comparisons are presented.*

Keywords: *Convolutional encoder, FPGA, verilog, Viterbi Algorithm, Viterbi decoder.*

I. Introduction

Convolutional coding is a popular error-correcting coding method used in digital communications. A message is convoluted, and then transmitted into a noisy channel. This convolution operation encodes some redundant information into the transmitted signal, thereby improving the data capacity of the channel. The Viterbi algorithm is a popular method used to decode convolutionally coded messages. The algorithm tracks down the most likely state sequences the encoder went through in encoding the message, and uses this information to determine the original message. Instead of estimating a message based on each individual sample in the signal, the convolution encoding and Viterbi decoding process packages and encodes a message as a sequence, providing a level of correlation between each sample in the signal.

A Viterbi decoder can be implemented using a FPGA [1] or an ASIC [2]. Implementing the Viterbi decoder using an ASIC is more efficient in terms of power and performance. However, an ASIC is, for the most part, a fixed design, and does not allow for much operational flexibility. A FPGA provides a large amount of operational flexibility, since the Viterbi algorithm is implemented as a program, which is executed by the FPGA. This flexibility is gained at the loss of performance and power efficiency. It is particularly suited to a channel in which the transmitted signal is corrupted mainly by additive white gaussian noise. Trellis coded modulation employs a high-rate Convolutional code as they are used in bandwidth-efficient systems. This leads to a high complexity of the Viterbi decoder for the TCM decoder [3], even if the constraint length of the Convolutional encoder is moderate. Due to the large number of transitions in the trellis diagram power consumption is more in Viterbi decoder. T-algorithm is an efficient technique to reduce the power consumption. However, searching for the optimal PM in the feedback loop still reduces the decoding speed. To overcome this drawback, two variations of the T-algorithm have been proposed [4]: the slow-down adaptive Viterbi decoding, which suggests using an estimated optimal path metric, instead of finding the real one each cycle and the limited-search parallel state Viterbi decoding based on scarce state transition (SST) [5]. Because of some drawbacks in both of them, an add-compare-select unit (ACSU) architecture based on pre-computation incorporating Viterbi algorithm for Viterbi decoders is proposed. This is efficiently improves Viterbi decoders clock speed reduce the power consumption.

II. Viterbi (Convolutional) Encoding and Viterbi Decoding.

2.1 Viterbi encoder:

Convolutional encoding of data is accomplished using a shift register and associated combinatorial logic that performs modulo-two addition. (A shift register is merely a chain of flip-flops wherein the output of the n^{th} flip-flop is tied to the input of the $(n+1)^{th}$ flip-flop. Every time the active edge of the clock occurs, the input to the flip-flop is clocked through to the output, and thus the data are shifted over one stage.) The combinatorial logic is often in the form of cascaded exclusive-or gates. As a reminder, exclusive-or gates are two-input, one-output gates often represented by the logic symbol shown in Fig.1.

The exclusive-or gate performs modulo-two addition of its inputs. When cascade of q two-input exclusive-or gates, with the output of the first one feeding one of the inputs of the second one, the output of the second one feeding one of the inputs of the third one, etc., the output of the last one in the chain is the modulo-two sum of the q + 1 inputs.

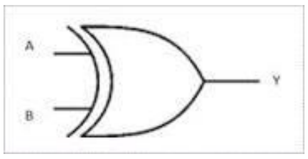

Fig.1: Ex-or gate

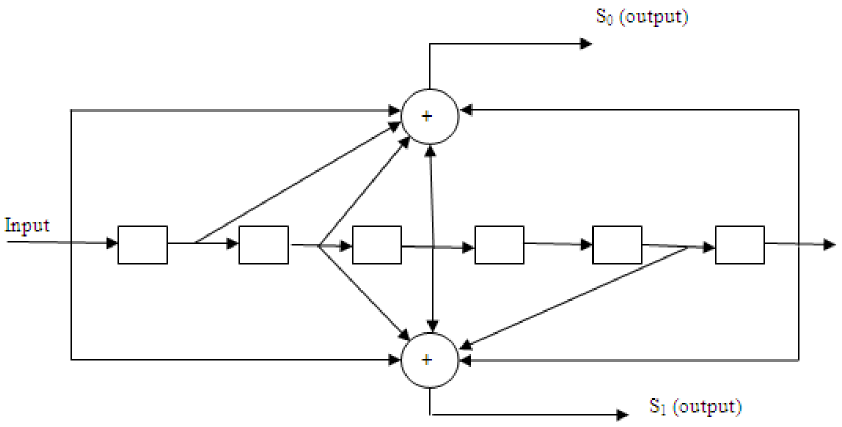

Fig.2: Encoder architecture with k=7 and k/n=1/2

Convolutional code is a type of error-correcting code which contains memory and ‘n’ encoder outputs at any given time unit depends not only on the k inputs at that time unit but also on m previous input blocks. Convolutional codes are usually characterized by two parameters code rate (k/n) and constraint length (K) and the patterns of ‘n’ modulo-2 adders. The shift register has a constraint length (K) of 7, equal to the number of stages in the register. The encoder has n generator polynomials, one for each adder. An input bit is fed into the leftmost register. Using the generator polynomials and the existing values in the remaining registers, the encoder generates output n bits. The code rate (k/n) is expressed as a ratio of the number of bits into the Convolutional encoder (k) to the number of channel symbols output by the Convolutional encoder (n) in a given encoder cycle. Convolutional encoder with constraint length 7 and code rate ½ is shown in the Fig.2.

2.2 Viterbi decoder:

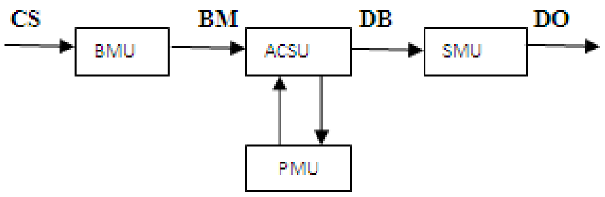

Fig. 3: Block diagram of viterbi decoder.

CS: CHANNEL SYMBOL
BM: BRANCH METRICS
DB: DECISION BITS

DO: DECODED OUTPUT
PM: PATH METRICS

Viterbi decoder block diagram is shown in Fig.3. Branch Metrics (BMs) are calculated in the BM unit (BMU) from the received symbols. Then, BMs are fed into the Add-Compare-Select Unit (ACSU) that recursively computes the Path Metrics (PM) and outputs decision bits for each possible state transition. After that, the decision bits are stored in and retrieved from the Survivor Metric Unit (SMU) in order to decode the source bits along the final survivor path. The PMs of the current iteration are stored in the PM unit (PMU). ACSU implementation is critical because the feedback loop makes it the bottleneck for high speed applications. Furthermore, as K value increases, the power consumption and computation complexity increase exponentially. In the T is set and the difference between each PM and the optimal one is calculated. So T-algorithm [4] requires extra computation in the ACSU loop for calculating the optimal PM (minimum value of all PMs) and puncturing states which was eliminated in proposed viterbi algorithm. Viterbi algorithm is mostly reliable for single bit error detection and correction. As number of error bits increases then efficiency of algorithm is decreases.

III. Proposed Viterbi Algorithm.

Viterbi algorithm is the most likelihood decoding algorithm of convolution code. In the area of communication, convolution code is very popular, so how to improve the performance and reduce the power and area of the decoder is important. On the other hand, different protocols use different convolution code and various applications have different requirement for throughput, area and power. So design of reusable Viterbi decoder is important. In the paper, a reusable Viterbi decoder is carried out. This decoder adopted the Process Element (PE) technique, which made it easy to adjust the throughput of the decoder by increasing or decreasing the number of PE. By the method of Same Address Write Back (SAWB), to reduce the number of registers to half in contrast with the method of Ping-pong.

3.1 Process Element

PE, Process Element, was one of the most popular architecture in digital signal process. The PE architecture of Viterbi decoder has been introduced in these papers [1] [2]. Fig. 4 is the basic structure of PE in Viterbi decoder, consists of PMU, ACSU, BMU and LRU. By assembling different numbers of PE, the PE appears state-serial, part parallel or full parallel structure of Viterbi decoder. And because the PMU is scattered into each PE, this structure is more area efficient than that structure of only one PE.

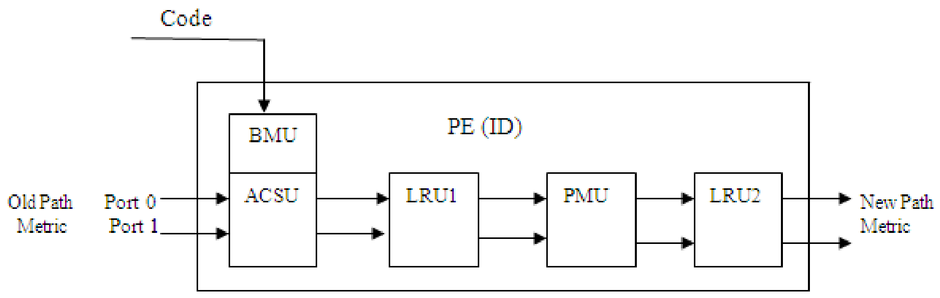

Fig. 4: Basic structure of PE in viterbi decoder.

3.2 Coordinate of States

In order to make two or more PE work together, we should send the appropriate state to the appropriate PE at the appropriate time. On this purpose, we can regard the binary presentation of each state as a coordinate and give special "meanings" to each bit. Look at such a binary presentation of state [2]

$$S = [X_u X_{u-1} \cdot \cdot X_1, Y_w Y_{w-1} \cdot \cdot Y_1, Z_v Z_{v-1} \cdot \cdot \cdot Z_1] \quad \mathbf{(1)}$$

$$Xi, Yi, Zi \in \{0, 1\}$$

$$u, w, v \in N, u + w + v = m$$

In (1) m is the depth of convolution coding. So, we define such a rule named "The Rule of PE" to explain the "meanings": The Rule of PE When $[X_u X_{u-1} \cdot\cdot\cdot X_1, Y_w Y_{w-1} \cdot\cdot\cdot Y_1, Z_v Z_{v-1} \cdot\cdot\cdot Z1]$ is the binary presentation of state S, S should be sent to the port $[Z_v Z_{v-1} \ldots Z_1]$ of PE $[Y_w Y_{w-1} \ldots Y_1]$ at time $[X_u X_{u-1} \ldots X_1]$.

Let Slice = $[X_uX_{u-1}\ldots X_1]$, Identity = $[Y_wY_{w-1}\ldots Y_1]$, Path = $[Z_vZ_{v-1}\ldots Z_1]$, so Slice, Identity and Path indicate the time of calculating path metric, the ID of Executing this calculation and the input port of PE respectively. u, v and w are three important parameters in the Rule of PE, because they determine the total slices for calculating(2^u),the radix of butterfly(2^v) and the total number of PE(2^w) respectively. Based on (1) and the Rule of PE, we can use binary presentation of state as the coordinate of it.

3.3 Same Address Write Back (S.A.W.B)

Usually there are two ways to design PMU: one is ping-pong method and the other is S.A.W.B. The later means write the new path metric back to where we read the old path metric from. It cost less storage unit than the method of ping-pong, which uses two same ram or register file to storage the new path metric and the old path metric respectively. By the method of S.A.W.B we reduce the storage units, but because it changes the storage address of path metric, we need some address transformation unit and more routing network. So being aware of the state flow graph in PE is very important when we adopt S.A.W.B method. Fig.5 demonstrates the state flow in PE.

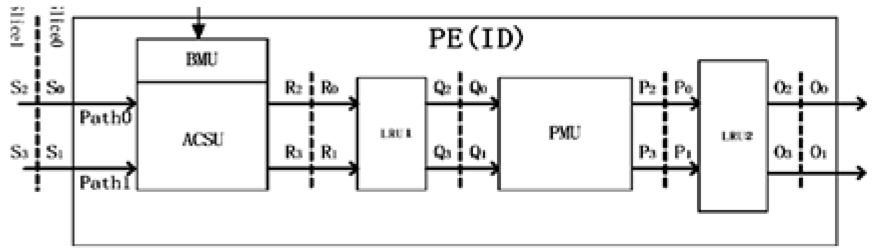

Fig. 5: State flow in pe.

In Fig.5 Path0 and Path1 are two in-ports of Radix-2 butterfly; ID is the number of PE; Slice0 and Slice1 are the taps. S_i, R_i, Q_i, P_i and O_i indicate the state at each in-port or code through the noise channel. Out-port of sub modules of PE respectively, i € {0, 1. . . 2^{m-1}}. Code is the convolution code through the noise channel. Firstly we define some operations:

$S_i = [X_uX_{u-1}\ldots X_1, Y_wY_{w-1}\ldots Y_1, Z_vZ_{v-1}\ldots Z_1]$

$\text{Shuffle}^v(S_i) = [Z_vZ_{v-1}\ldots Z_1, X_uX_{u-1}\ldots X_1, Y_wY_{w-1}\ldots Y_1]$

$\Gamma^u_{u+v}(S_i) = [\text{Shuffle}^v(X_uX_{u-1}\ldots X_1,, Y_wY_{w-1}\ldots Y1), Z_vZ_{v-1}\ldots Z_1]$

$= [Y_vY_{v-1}\ldots Y_1X_uX_{u-1}\ldots X_{v+1}, X_vX_{v-1}\ldots X_1Y_wY_{w-1}\ldots Y_{v+1}, Z_vZ_{v-1}\ldots Z_1]$

Because of the ACS operation in ACSU, R_i is not the same of S_i and there exists a transformation between S_i and R_i as following:

$R_i = \text{Shuffle}^v(S_i) = [Z_vZ_{v-1}\ldots Z1, X_uX_{u-1}\ldots X_1, Y_wY_{w-1}\ldots Y_i]$ **(2)**

(2) Is determined by the ACS operation in ACSU. From it we can see the least w bits of R_i are always equal to $[Y_wY_{w-1}\ldots Y_1]$, the ID of PE.

Let's look at state O_i and leave Q_i and Pi for later. O_i is transformed from R_i through LRU1, PMU and LRU2. The least w bits of O_i should be $[Y_wY_{w-1}\ldots Y1]$, supposing that LRU1, PMU and LRU2 only reschedule the sequence of state flow and not change the set of states. Because the out-ports of PE are connected to certain in-ports of another PE, O_i should be certain S_i at certain in-port of certain PE with the same slice. That indicates the most u bits of O_i should be $[X_uX_{u-1}\ldots X1]$, the slice of S_i. Any more, O_i is rearranged from R_i and so the set of O_i and the set of R_i should be same. All above, we appoint such value to O_i:

$O_i = [X_uX_{u-1}\ldots X_1, Z_vZ_{v-1}\ldots Z_1, Y_w Y_{w-1}\ldots Y_1]$ **(3)**

$= [\text{Shuffle}^u(Z_vZ_{v-1}\ldots Z_1, X_uX_{u-1}\ldots X_1), Y_wY_{w-1}\ldots Y_1]$

$= \Gamma^u_{u+v}(Z_vZ_{v-1}\ldots Z_1, X_uX_{u-1}\ldots X_1, Y_wY_{w-1}\ldots Y_1)$

(3) Meets all the requirements described above. Maybe there exists some other appointment of Q_i, but (3) maybe the simplest. Further more if $< S >_j$ denotes the address of state S at cycle j (cycle is the time during which we calculate the path metric of all states), then $< R_i >_{j+1} = < O_i >_j$ is directly derived from the define of S.A.W.B. And based on (2), (3) we can write out the address transformation caused by S.A.W.B:

$< S >_{j+1} = \text{S.A.W.B}(< S >_j) = < \Gamma^u_{u+v}(S) >_j$ **(4)**

Finally, Q_i and P_i are determined by R_I and O_i respectively. If there are no LRU1 or LRU2, Q_i and P_i will be equal to R_i and O_i respectively. Because LRU1 or LRU2 is not a storage unit, it rerouting the in-ports into out-ports, how to choice the value of Q_i and Pi is determined by what structure of the PMU. We will determine the value of Q_i and P_i in the next section.

3.4 Division of states

In PMU, register file used for path-metric storage. If use only one register file for read and write, we will need a register file with two read and two write ports for radix-2 butterfly. But this kind of register file is very complicated and not area-efficient, so we decide to divide the set of all states calculated by each PE into different subsets. If we use Radix-2^v butterfly, we will divide these states into 2^v subsets and storage them the same number of register files. On this purpose, we should distinguish between the Q_i with the same slice and between the P_i with the same slice, too. In other words, if we use function Θ to distinguish between different Q_i and between different P_i, it should satisfied this:

Class Rule: ¥ Q_j, Q_k €{Q_i|Q_i with the same slice } and $Q_j \neq Q_k$, we have $\Theta(Q_j) \neq \Theta(Q_k)$; ¥ P_j , P_k € {P_i|P_i with the same slice} and $P_j \neq P_k$, we have $\Theta(P_j) \neq \Theta(P_k)$, too. In order to make problem simple, we provide that u is multiple of v. It is not difficult to be satisfied for v = 1. With provision of this, we divide the binary presentation of state into u/v groups each having v bits. Then we sum these u/v groups with mod-2^v-add. From the sum we could divide set {Q} or {P} into 2^v subsets. It is not hard to be validated. So LRU1 reroutes R_i into Q_i and LRU2 reroutes P_i into O_i with the right order which is determined by the Class Rule.

3.5 Viterbi Decoder Analysis

The viterbi decoder can be analyzed in three steps.

STEP 1: Weigh the trellis; that is, calculate the branch metrics.

STEP 2: Recursively computes the shortest paths to time n, in terms of the shortest paths to time n-1. In this step, decisions are used to recursively update the survivor path of the signal. This is known as add-compare-select (ACS) recursion.

STEP 3: Recursively finds the shortest path leading to each trellis state using the decisions from Step 2. The shortest path is called the survivor path for that state and the process is referred to as survivor path decode. Finally, if all survivor paths are traced back in time, they merge into a unique path, which is the most likely signal path.

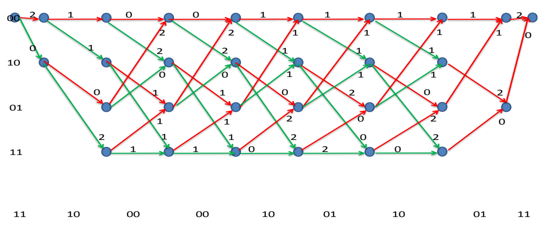

Fig.: 6 decoding for calculation of hamming distance.

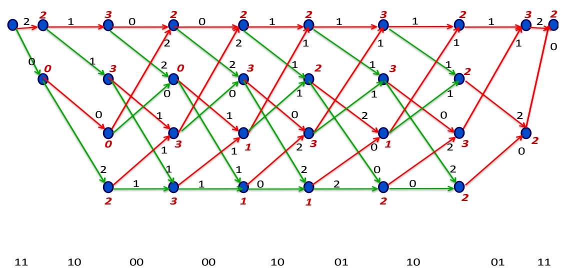

Fig.: 7 decoding for calculation of path metric.

Fig. 7 shows the decoding sequence for two code words received. The hamming distance between the code words being received and the output of encoder is calculated.

Consider the third code word being received. The path metrics are simply the addition of branch metric and previous path metric. For state (00), the partial path metric through the line marked 00 is 2, while the partial path metric through the line marked 10 is 3. The former is less than the later. So the survivor path of state (00) is through the solid line and path metric for the state is updated as 2. The same operations are done for each state.

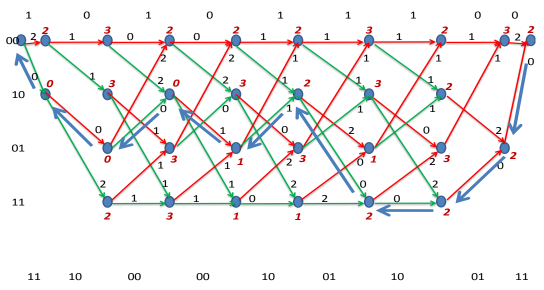

Fig.: 8 Decoding for noisy channel using trace back method.

When the computation of path metric is done, the next step is to decode the most similar sequence. The decoding process is denoted as trace back because this process is like tracing the sequence back. Here it traces the best path from the state with the minimum path metric. The final path is highlighted in Fig. 8. The Fig. 8 shows that when the final path is traced back even with one bit of error, output is correct.

IV. Simulation and Synthesis Results of Viterbi Decoder.

The RTL module of Viterbi Encoder and Viterbi Decoder are shown in Fig. 9 and Fig. 10 respectively.

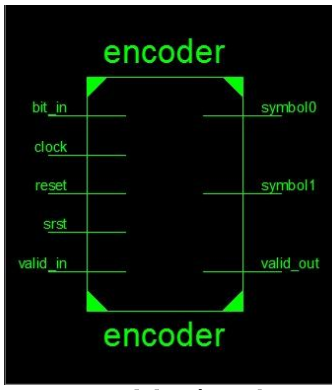

Fig.:9 RTL module of viterbi encoder

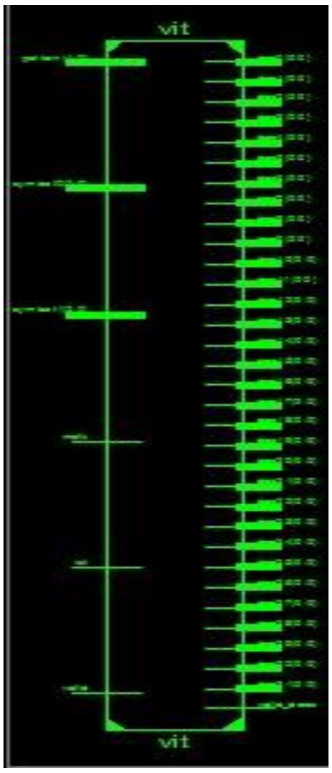

Fig.:10 RTL module of viterbi decoder

Viterbi decoder with rate ½ and K=7 is implemented on Spartan 3E FPGA device XC3S500E-4FG320 and Spartan 6 FPGA device XC6SLX4-3TQG144. The simulation wave forms are shown in Fig.11. The pre computation Viterbi decoder reduced the power consumption by nearly 47% with minimum decoding speed at 103.096MHz.The Viterbi decoder design is Simulated and Synthesized in Xilinx ISE 12.1.

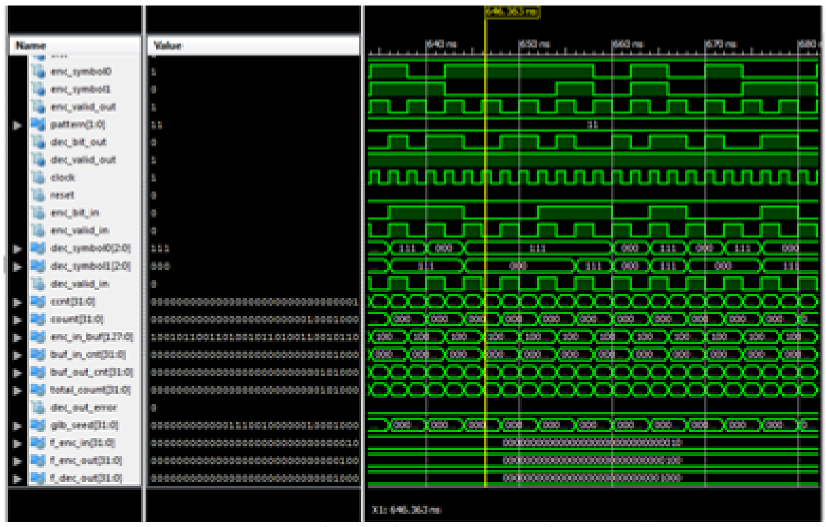

Fig. 11: Simulation wave forms for decoding of Convolutional codes.

Compared with the full-trellis VD without a low-power scheme, the VD could have low power consumption with reliable decoding speed as shown in Table I. A reusable Viterbi decoder was carried out by adopting the Process Element technique.

Table: I Power Estimation Results

	Power (mW)	
Full-trellis VD	21.473	
VD with 2-step pre-computation	T_{pm} = 0.75	20.069
	T_{pm} = 0.625	17.186
	T_{pm} = 0.5	11.754
	T_{pm} = 0.375	6.6127
Viterbi algorithm	38.84	

The device utilization summary, logic utilization and distribution report of Spartan 3E based XC3S500E-4FG320 FPGA device is shown in Table II (a) and power analysis report is shown in Table II (b).

SPARTAN 3E

Table: II (a) Synthesis and PAR report

Parameters	USED	AVAILABLE	UTILIZATION (%)
Number of slice FF	696	9312	7
Number of occupied slices	1005	4656	21
Number of bonded IOBs	14	232	6
Number of BRAM	2	20	10
Number of BUFGMUXs	1	24	4

Table: II (b) Power analysis report

Frequency (MHz)	Dynamic power(W)	Leakage power(W)	Total power(W)
66.378	0.028	0.081	0.103
50	0.022	0.081	0.125
30	0.014	0.081	0.095
20	0.010	0.081	0.091

The device utilization summary, logic utilization and distribution report of Spartan 6 based XC6SLX4-3TQG144 FPGA device is shown in Table II (a) and power analysis report is shown in Table II (b).

Spartan 6

Table: II (a) Synthesis and PAR report

Parameters	USED	AVAILABLE	UTILIZATION (%)
Number of slice registers	698	4800	14
Number of slice LUTs	1922	2400	80
Number of fully used LUT-FF	660	1924	34
Number of bonded IOBs	14	102	13
Number of BRAMs	2	12	16
Number of BUFG/BUFGMUX	1	16	6

Table: II (b) Power analysis report

FREQUENCY(MHz)	Dynamic power(W)	Leakage power(W)	Total power(W)
103.096	0.025	0.014	0.039
100	0.024	0.014	0.038
50	0.013	0.014	0.026
20	0.005	0.014	0.019

The power and area analysis report of Spartan 3E based XC3S500E-4FG320 FPGA device and Spartan 6 based XC6SLX4-3TQG144 device are shown in Table III.

Table: III Power and Area analysis report

Performance Metrics	FPGA IMPLEMENTATION (XILINX SPARTAN 3E)	FPGA IMPLEMENTATION (XILINX SPARTAN 6)
DYNAMIC POWER	28mW	25mW
LEAKAGE POWER	81mW	14mW
AREA (in terms of LUTs)	1812 LOGIC CELLS	1922 LOGIC CELLS
TOTAL NO OF REGISTERS	696	698

V. Conclusion

A Viterbi Decoder can be implemented using a FPGA or an ASIC. Implementing the Viterbi decoder using ASIC is more efficient in terms of power and performance. However, an ASIC is, for the most part, a fixed design, and does not allow for much operational flexibility. A FPGA furnishes a large amount of operational flexibility, since the Viterbi algorithm is implemented as a program, which is executed by the FPGA. This flexibility is acquired at the loss of performance and power efficiency. It is particularly suited to a channel in which the transmitted signal is corrupted mainly by Additive White Gaussian Noise. Viterbi decoder with rate ½ and K=7 is implemented in Spartan 3E based XC3S500E-4FG320 FPGA device and Spartan 6 based XC6SLX4-3TQG144 FPGA device. The pre computation Viterbi decoder reduced the power consumption by nearly 47% with minimum decoding speed at 103.096 MHz. The Viterbi Decoder design is simulated and synthesized in Xilinx ISE 12.1. The results show that proposed design can work at frequency of 103.096 MHz by using considerable less resources of target FPGA to provide high performance, cost effective solution for wireless communication applications.

References

[1] Montse Boo, Francisco Arguello, Javier D.Bruguera, Ramon Doallo, and Emilio L.Zapata." High-Performance VLSI Architecture for the Viterbi Algorithm". IEEE Trans. Communications, 45(2), February 1997.

[2] F.Arguello, J.D.Bruguera, R.Doallo, and E.L.Zapata. "Parallel Architecture for Fast Transforms with Trigonometric Kernel". IEEE Trans. Parallel and Distributed Systems, 5(10), October 1994.

[3] J. He, Z. Wang, and H. Liu, "An efficient 4-D 8PSK TCM decoder architecture," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 18, no. 5, pp. 808–817, May 2010.

[4] J. He, H. Liu, and Z. Wang, "A fast ACSU architecture for viterbi decoder using T-algorithm," in Proc. 43rd IEEE Asilomar Conf. Signals,Syst. Comput., Nov. 2009, pp. 231–235.

[5] Jie Jin, Chi-ying Tsui "Parallel state Viterbi decoder implementation based on scarce state transition," IEEE Trans.Very Large Scale Integr. (VLSI) Syst., vol. 15, no. 11, pp. 1172–1176,Oct. 2007.

[6] J. He, H. Liu, Z. Wang,X.Huang and Kai Zhang ,"High-Speed Low-Power Viterbi Decoder Design for TCM Decoders" in IEEE Trans.on (VLSI) Systems, Vol. 20, No. 4, April 2012.