IMPLEMENTATION OF VITERBI DECODER WITH VARIABLE CODE RATE

T.VENU GOPAL

[1]Associate Professor, ECE, Vidya jyothi Institute of Technology,aziz nagar,c.b.post

ABSTRACT: ***It is well known that data transmissions over wireless channels are affected by attenuation, distortion, interference and noise, which affect the receiver's ability to receive correct information. Convolution encoding with VITERBI decoding is a good forward error correction technique suitable for channels affected by noise degradation. It has been widely deployed in many wireless communication systems to improve the limited capacity of the communication channels. A new efficient fangled VITERBI algorithm is proposed in this paper with less complexity and processing time along with 2 bit error correction capabilities and variable code rate. The design is successfully simulated using Xilinx ISIM and targeted to xc3s500e FPGA device.***

KEYWORDS: VITERBI, xc3s500e FPGA device, attenuation, distortion, interference and noise

1. Introduction:

A Viterbi decoder uses the Viterbi algorithm for decoding a bitstream but here we are using this viterbi decoder for decoding of continuous bit stream that has been encoded using Forward error correction based on a Convolutional code. various type of decoders are used for decoding of Convolutional code but viterbi decoder is the best way to decode Convolutional code and it will give the error free original input at the receiver compared to other decoders.The aim this paper is to implement an efficient

Viterbi decoder with variable code rate i.e it means the decoder is used for different code rates for example ½ or 2/3. But in this project the viterbi decoder is desined for both ½ and 2/3 code rates depending on the input. Here there is interface between the ½ and 2/3 code rate.It is well known that data transmissions over wireless channels are affected by attenuation, distortion, interference and noise, which affect the receiver's ability to receive correct information. Convolution encoding with Viterbi decoding is a good forward error correction technique suitable for channels affected by noise degradation. A new efficient fangled Viterbi algorithm is proposed in this paper with less complexity and processing time along with 2 bit error correction capabilities and variable code rate. The design is successfully simulated using Xilinx ISIM and targeted to xc3s500e FPGA device.

Convolutional codes are commonly described using two parameters: the code rate and the constraint length. 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. The constraint length parameter, K, denotes the "length" of the Convolutional encoder, i.e. how many k-bit stages are available to feed the combinatorial logic that produces the output symbols. Closely related to K is the parameter m, which indicates how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the Convolution encoder. The m

ISSN: 2278 – 1323

parameter can be thought of as the memory length of the encoder. Convolution codes are widely used as channel codes in practical communication systems for error correction. The encoded bits depend on the current k input bits and a few past input bits.

The main decoding strategy for Convolution codes is based on the widely used Viterbi algorithm. As a result of the wide acceptance of Convolution codes, there have been several approaches to modify and extend this basic coding scheme. Trellis coded modulation (TCM) and turbo codes are two such examples. In TCM, redundancy is added by combining coding and modulation into a single operation. This is achieved without any reduction in data rate or expansion in bandwidth as required by only error correcting coding schemes.

Convolution encoding is considered to be one of the forward error correction scheme. This coding scheme is often used in the field of deep space communications and more recently in digital wireless communications. Adaptive Viterbi decoders are used to decode Convolution codes. This technique is used for error correction. It is very efficient and robust. The main advantage of Viterbi Decoder is it has fixed decoding time and also it suites for hardware decoding implementation. But the implementation requires the exponential increase in the area and power consumption to achieve increased decoding accuracy. Convolution coding with Viterbi decoding is a FEC technique that is particularly suited to a channel in which transmitted signal is corrupted mainly by additive white Gaussian noise (AWGN). In most of real time applications like audio and video applications, the Convolution codes are used for error correction. Most of the Viterbi decoders in the market are a parameterizable intelligent property (IP) core with an efficient algorithm for decoding of one convolution ally-encoded sequence only and this is mostly used in the field of satellite and radio communications. In addition, the cost for the Convolution encoder and Viterbi decoder is expensive for a specified design because of the patent issue. Therefore, to realize an adaptive Convolution encoder and Viterbi decoder on a field programmable gate array (FPGA) board is very demanding. In this paper, we concern with designing and implementing a Convolution encoder and Viterbi decoder which are the essential block in digital communication systems using FPGA technology.

Convolution codes offer an alternative to block codes for transmission over a noisy channel. Convolution coding can be applied to a continuous input stream (which cannot be done with block codes), as well as blocks of data. The overall block diagram of this paper is shown in the fig. 1. The message bits are generated and they are sent to the crypto system and then the decoded output is obtained. The type of crypto system used is Convolution Encoder and Viterbi Decoder. The Convolutional encoder encodes the message and then the encoded bits are generated. The bits which are encoded are again sent to the Viterbi Decoder and then the A Viterbi decoder uses the Viterbi algorithm for decoding a bitstream but here we are using this viterbi decoder for decoding of continuous bit stream that has been encoded using Forward error correction based on a Convolution code. various type of decoders are used for decoding of Convolutional code but viterbi decoder is the best way to decode Convolutional code and it will give the error free original input at the receiver compared to other decoders.

The aim this paper is to implement an efficient Viterbi decoder with variable code rate i.e it means the decoder is used for different code rates for example ½ or 2/3. But in this project the viterbi decoder is desined for both ½ and 2/3 code rates depending on the input. Here there is interface between the ½ and 2/3 code rate.

ISSN: 2278 – 1323

It is well known that data transmissions over wireless channels are affected by attenuation, distortion, interference and noise, which affect the receiver's ability to receive correct information. Convolution encoding with Viterbi decoding is a good forward error correction technique suitable for channels affected by noise degradation. A new efficient fangled Viterbi algorithm is proposed in this paper with less complexity and processing time along with 2 bit error correction capabilities and variable code rate. The design is successfully simulated using Xilinx ISIM and targeted to xc3s500e FPGA device.Convolutional codes are commonly described using two parameters: the code rate and the constraint length. The code rate, k/n, is expressed as a ratio of the number of bits into the Convolution encoder (k) to the number of channel symbols output by the Convolution encoder (n) in a given encoder cycle. The constraint length parameter, K, denotes the "length" of the Convolutional encoder, i.e. how many k-bit stages are available to feed the combinatorial logic that produces the output symbols. Closely related to K is the parameter m, which indicates how many encoder cycles an input bit is retained and used for encoding after it first appears at the input to the Convolution encoder. The m parameter can be thought of as the memory length of the encoder.

Convolutional codes are widely used as channel codes in practical communication systems for error correction. The encoded bits depend on the current k input bits and a few past input bits. The main decoding strategy for Convolution codes is based on the widely used Viterbi algorithm. As a result of the wide acceptance of Convolution codes, there have been several approaches to modify and extend this basic coding scheme. Trellis coded modulation (TCM) and turbo codes are two such examples. In TCM, redundancy is added by combining coding and modulation into a single operation. This is achieved without any reduction in data rate or expansion in bandwidth as required by only error correcting coding schemes. Convolution encoding is considered to be one of the forward error correction scheme. This coding scheme is often used in the field of deep space communications and more recently in digital wireless communications. Adaptive Viterbi decoders are used to decode Convolution codes.

This technique is used for error correction. It is very efficient and robust. The main advantage of Viterbi Decoder is it has fixed decoding time and also it suites for hardware decoding implementation. But the implementation requires the exponential increase in the area and power consumption to achieve increased decoding accuracy. Convolution coding with Viterbi decoding is a FEC technique that is particularly suited to a channel in which transmitted signal is corrupted mainly by additive white Gaussian noise (AWGN). In most of real time applications like audio and video applications, the Convolutional codes are used for error correction.

Most of the Viterbi decoders in the market are a parameterizable intelligent property (IP) core with an efficient algorithm for decoding of one convolution ally-encoded sequence only and this is mostly used in the field of satellite and radio communications. In addition, the cost for the Convolutional encoder and Viterbi decoder is expensive for a specified design because of the patent issue. Therefore, to realize an adaptive Convolutional encoder and Viterbi decoder on a field programmable gate array (FPGA) board is very demanding. In this paper, we concern with designing and implementing a Convolutional encoder and Viterbi decoder which are the essential

ISSN: 2278 – 1323

block in digital communication systems using FPGA technology.

Convolutional codes offer an alternative to block codes for transmission over a noisy channel. Convolutional coding can be applied to a continuous input stream (which cannot be done with block codes), as well as blocks of data. The overall block diagram of this paper is shown in the fig. 1. The message bits are generated and they are sent to the crypto system and then the decoded output is obtained. The type of crypto system used is Convolutional Encoder and Viterbi Decoder. The Convolutional encoder encodes the message and then the encoded bits are generated. The bits which are encoded are again sent to the Viterbi Decoder and then the decoded output is obtained.

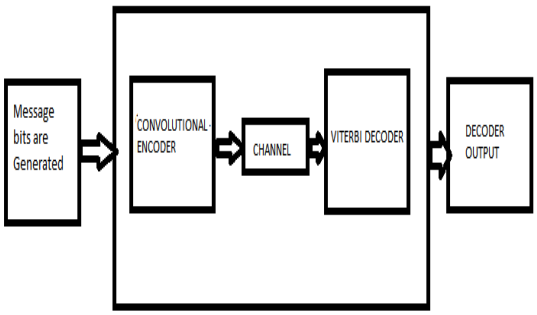


Figure 1: Convolutional encoding and viterbi decoding Block diagram

The following terms are vital to the understanding of Convolutional coding and viterbi decoding.

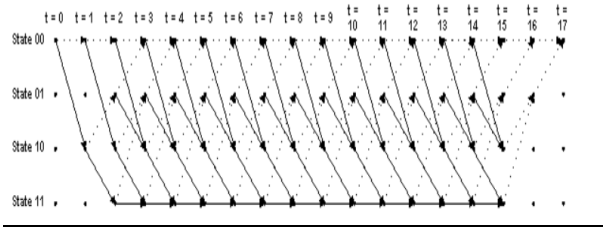
1. **Hard-decision/soft-decision decoding**: Hard-decision decoding means that the demodulator is quantized to two levels: zero and one. If you derive more than two quantization levels from the demodulator, then the decoder is soft-decision decoding.
2. **Code rate R(=k/n):** Number of bits into Convolutional encoder (k)/ number of bits in output symbol which corresponds not only current input bit, but also previous (K – 1) ones. It is also defined as the ratio of the number of output bits to the number of input bits.
3. **Constraint length K**: It denotes the "length" of the Convolutional encoder, i.e., no of k bit stages are available to feed the combinatorial logic that produces the output symbols.
4. **Branch Metric**: Difference between the received sequence and the branch word is called the Branch metric.
5. **Path Metric**: Branch metric accumulates to form Path metric.
6. **Hard-decision/soft-decision decoding**: Hard-decision decoding means that the demodulator is quantized to two levels: zero and one. If you derive more than two quantization levels from the demodulator, then the decoder is soft-decision decoding.
7. **Code rate R(=k/n):** Number of bits into Convolutional encoder (k)/ number of bits in output symbol which corresponds not only current input bit, but also previous (K – 1) ones. It is also defined as the ratio of the number of output bits to the number of input bits.
8. **Constraint length K**: It denotes the "length" of the Convolutional encoder, i.e., no of k bit stages are available to feed the combinatorial logic that produces the output symbols.
9. **Branch Metric**: Difference between the received sequence and the branch word is called the Branch metric.
10. **Path Metric**: Branch metric accumulates to form Path metric.

3 Implementation:

3.1 Performing Viterbi Decoding:

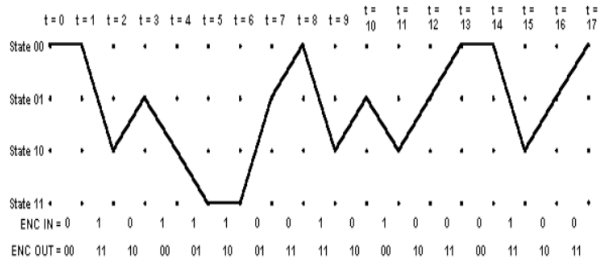
ISSN: 2278 – 1323

The Viterbi decoder itself is the primary focus of this tutorial. Perhaps the single most important concept to aid in understanding the Viterbi algorithm is the trellis diagram. The figure below shows the trellis diagram for our example rate 1/2 K = 3 Convolutional encoder, for a 15-bit message:

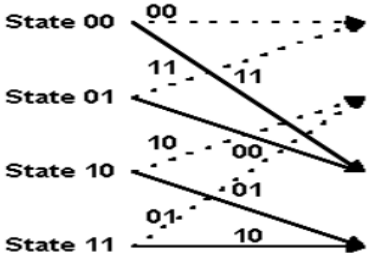


The four possible states of the encoder are depicted as four rows of horizontal dots. There is one column of four dots for the initial state of the encoder and one for each time instant during the message. For a 15-bit message with two encoder memory flushing bits, there are 17 time instants in addition to t = 0, which represents the initial condition of the encoder. The solid lines connecting dots in the diagram represent state transitions when the input bit is a one. The dotted lines represent state transitions when the input bit is a zero. Notice the correspondence between the arrows in the trellis diagram and the state transition table discussed above. Also notice that since the initial condition of the encoder is State 002, and the two memory flushing bits are zeroes, the arrows start out at State 002 and end up at the same state.

The following diagram shows the states of the trellis that are actually reached during the encoding of our example 15-bit message:



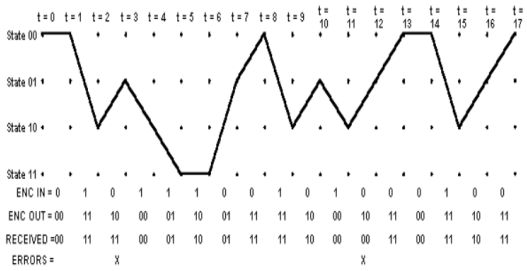
The encoder input bits and output symbols are shown at the bottom of the diagram. Notice the correspondence between the encoder output symbols and the output table discussed above. Let's look at that in more detail, using the expanded version of the transition between one time instant to the next shown below:



The two-bit numbers labelling the lines are the corresponding convolutional encoder channel symbol outputs. Remember that dotted lines represent cases where the encoder input is a zero, and solid lines represent cases where the encoder input is a one. (In the figure above, the two-bit binary numbers labelling dotted lines are on the left, and the two-bit binary numbers labelling solid lines are on the right.)

OK, now let's start looking at how the Viterbi decoding algorithm actually works. For our example, we're going to use hard-decision symbol inputs to keep things simple. (The example source code uses soft-decision inputs to achieve better performance.) Suppose we receive the above encoded message with a couple of bit errors:

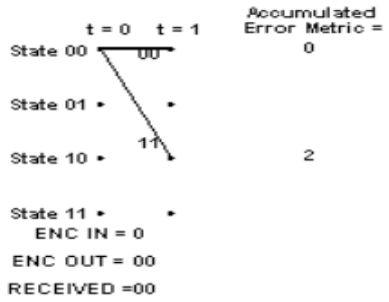
ISSN: 2278 – 1323



Each time we receive a pair of channel symbols, we're going to compute a metric to measure the "distance" between what we received and all of the possible channel symbol pairs we could have received. Going from t = 0 to t = 1, there are only two possible channel symbol pairs we could have received: 002, and 112. That's because we know the Convolutional encoder was initialized to the all-zeroes state, and given one input bit = one or zero, there are only two states we could transition to and two possible outputs of the encoder. These possible outputs of the encoder are 00 2 and 112.

The metric we're going to use for now is the Hamming distance between the received channel symbol pair and the possible channel symbol pairs. The Hamming distance is computed by simply counting how many bits are different between the received channel symbol pair and the possible channel symbol pairs. The results can only be zero, one, or two. The Hamming distance (or other metric) values we compute at each time instant for the paths between the states at the previous time instant and the states at the current time instant are called branch metrics. For the first time instant, we're going to save these results as "accumulated error metric" values, associated with states. For the second time instant on, the accumulated error metrics will be computed by adding the previous accumulated error metrics to the current branch metrics.

At t = 1, we received 002. The only possible channel symbol pairs we could have received are 002 and 112. The Hamming distance between 002 and 002 is zero. The Hamming distance between 002 and 112 is two. Therefore, the branch metric value for the branch from State 002 to State 002 is zero, and for the branch from State 002 to State 102 it's two. Since the previous accumulated error metric values are equal to zero, the accumulated metric values for State 002 and for State 102 are equal to the branch metric values. The accumulated error metric values for the other two states are undefined. The figure below illustrates the results at t = 1:

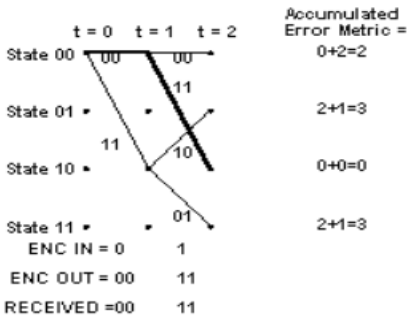


Note that the solid lines between states at t = 1 and the state at t = 0 illustrate the predecessor-successor relationship between the states at t = 1 and the state at t = 0 respectively. This information is shown graphically in the figure, but is stored numerically in the actual implementation. To be more specific, or maybe clear is a better word, at each time instant t, we will store the number of the predecessor state that led to each of the current states at t.

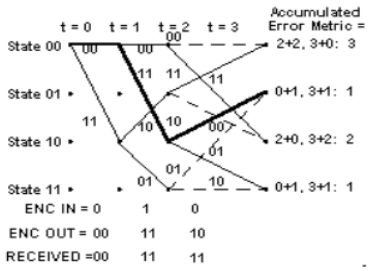
Now let's look what happens at t = 2. We received a 112 channel symbol pair. The possible channel symbol pairs we could have received in going from t = 1 to t = 2 are 002 going from State 002 to State 002, 112 going from State 002 to State 102, 102 going from State 102 to State 01 2, and 012 going from State 102 to State 11 2. The Hamming distance between 002 and 112 is two, between 112 and 112 is zero, and between 10 2 or 012 and 112 is one. We add these branch metric values to the previous

ISSN: 2278 – 1323

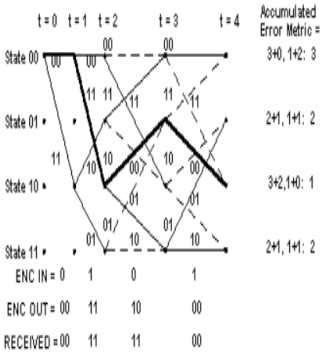
accumulated error metric values associated with each state that we came from to get to the current states. At t = 1, we could only be at State 002 or State 102. The accumulated error metric values associated with those states were 0 and 2 respectively. The figure below shows the calculation of the accumulated error metric associated with each state, at t = 2.



That's all the computation for t = 2. What we carry forward to t = 3 will be the accumulated error metrics for each state, and the predecessor states for each of the four states at t = 2, corresponding to the state relationships shown by the solid lines in the illustration of the trellis.Now look at the figure for t = 3. Things get a bit more complicated here, since there are now two different ways that we could get from each of the four states that were valid at t = 2 to the four states that are valid at t = 3. So how do we handle that? The answer is, we compare the accumulated error metrics associated with each branch, and discard the larger one of each pair of branches leading into a given state. If the members of a pair of accumulated error metrics going into a particular state are equal, we just save that value. The other thing that's affected is the predecessor-successor history we're keeping. For each state, the predecessor that survives is the one with the lower branch metric. If the two accumulated error metrics are equal, some people use a fair coin toss to choose the surviving predecessor state. Others simply pick one of them consistently, i.e. the upper branch or the lower branch. It probably doesn't matter which method you use. The operation of adding the previous accumulated error metrics to the new branch metrics, comparing the results, and selecting the smaller (smallest) accumulated error metric to be retained for the next time instant is called the add-compare-select operation. The figure below shows the results of processing t = 3:

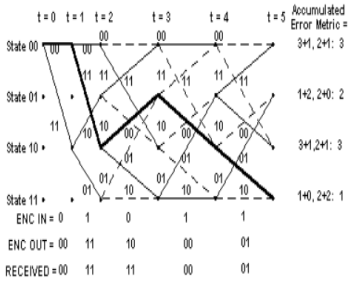


Note that the third channel symbol pair we received had a one-symbol error. The smallest accumulated error metric is a one, and there are two of these. Let's see what happens now at t = 4. The processing is the same as it was for t = 3. The results are shown in the figure:



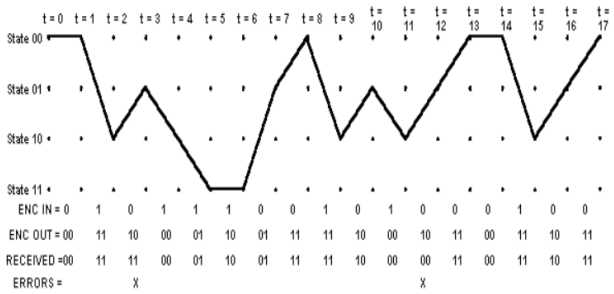
Notice that at t = 4, the path through the trellis of the actual transmitted message, shown in bold, is again associated with the smallest accumulated error metric. Let's look at t = 5:

ISSN: 2278 – 1323



At t = 5, the path through the trellis corresponding to the actual message, shown in bold, is still associated with the smallest accumulated error metric. This is the thing that the Viterbi decoder exploits to recover the original message.

At t = 17, the trellis looks like this (clutter of the intermediate state history removed):



The decoding process begins with building the accumulated error metric for some number of received channel symbol pairs, and the history of what states preceded the states at each time instant t with the smallest accumulated error metric. Once this information is built up, the Viterbi decoder is ready to recreate the sequence of bits that were input to the convolutional encoder when the message was encoded for transmission. This is accomplished by the following steps:

1) First, select the initial state as the one having the smallest accumulated error metric and save the state number of that state.

2) Iteratively perform the following step until the beginning of the trellis is reached: Working backward through the state history table, for the currently selected state, select a new state by looking in the state history table for the predecessor to the current state. Save this state number as the new currently selected state and continue. This step is called traceback.

3) Now work forward through the list of selected states saved in the previous steps. Look up what input bit corresponds to a transition from each predecessor state to its successor state. That is the bit that must have been encoded by the convolutional encoder.

The following table shows the accumulated metric for the full 15-bit (plus two flushing bits) example message at each time t. We will use this table only to select the initial, first state for decoding:

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State 00_2		0	2	3	3	3	3	4	1	3	4	3	3	2	2	4	5	2
State 01_2			3	1	2	2	3	1	4	4	1	4	2	3	4	4	2	
State 10_2		2	0	2	1	3	3	4	3	1	4	1	4	3	3	2		
State 11_2			3	1	2	1	1	3	4	4	3	4	2	3	4	4		

It is interesting to note that for this hard-decision-input Viterbi decoder example, the smallest accumulated error metric in the final state indicates how many channel symbol errors occurred. From this table, we can see that the initial, first state to select is state '0'.

The following state history table shows the surviving predecessor states for each state at each time t. We will use this table to perform the actual step-by-step traceback:

ISSN: 2278 – 1323

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State 00_2	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	1
State 01_2	0	0	2	2	3	3	2	3	3	2	2	3	2	3	2	2	2	0
State 10_2	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0
State 11_2	0	0	2	2	3	2	3	2	3	2	2	3	2	3	2	2	0	0

The following table shows the states selected when tracing the path back through the survivor state table shown above:

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0	2	1	0

Using a table that maps state transitions to the inputs that caused them, we can now recreate the original message. Here is what this table looks like for our example rate 1/2 K = 3 Convolutional code:

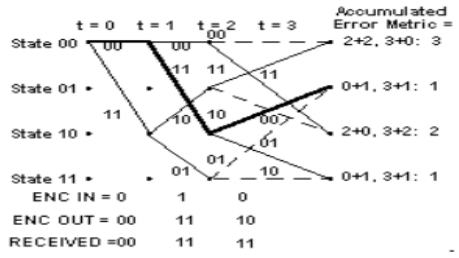
	Input was, Given Next State =			
Current State	**00_2 = 0**	**01_2 = 1**	**10_2 = 2**	**11_2**
00_2 = 0	0	x	1	x
01_2 = 1	0	x	1	x
10_2 = 2	x	0	x	1
11_2 = 3	x	0	x	1

Note: In the above table, x denotes an impossible transition from one state to another state. So now we have all the tools required to recreate the original message from the message we received:

t =	1	2	3	4	5	6	7	8	9	10	11	12	13	1
	0	1	0	1	1	1	0	0	1	0	1	0	0	

The two flushing bits are discarded.

Here's an insight into how the traceback algorithm eventually finds its way onto the right path even if it started out choosing the wrong initial state. This could happen if more than one state had the smallest accumulated error metric, for example. I'll use the figure for the trellis at t = 3 again to illustrate this point:



See how at t = 3, both States 012 and 112 had an accumulated error metric of 1. The correct path goes to State 012 -notice that the bold line showing the actual message path goes into this state. But suppose we choose State 112 to start our trace back. The predecessor state for State 112 , which is State 102 , is the same as the predecessor state for State 012! This is because at t = 2, State 102 had the smallest accumulated error metric. So after a false start, we are almost immediately back on the correct path.

4. Simulation Of Viterbi Decoder:

4.1 RTL Schematic diagram of Convolutional encoder:

The below RTL schematic diagram for the code rte of both k=1/2 and k=2/3 is as follows

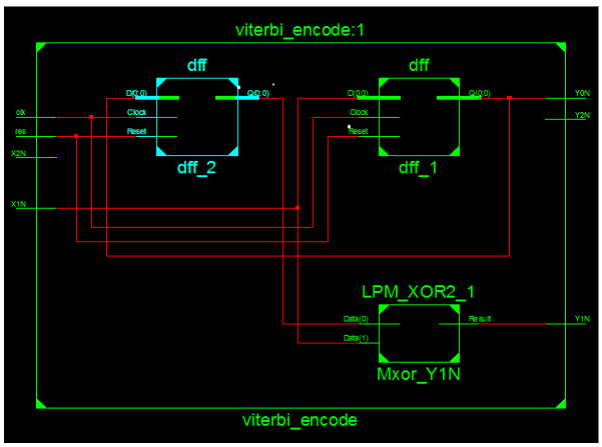


Figure 4.1: RTL Schematic of Convolutional encoder

ISSN: 2278 – 1323

4.2 Block diagram & RTL Schematic Diagram of Viterbi Decoder:

The block diagram and the RTL schematic diagram of viterbi decoder is as follows

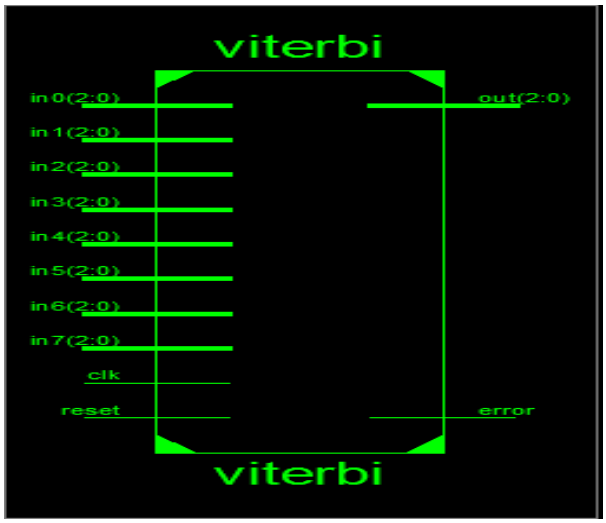


Figure 4.2a: Block diagram viterbi decoder

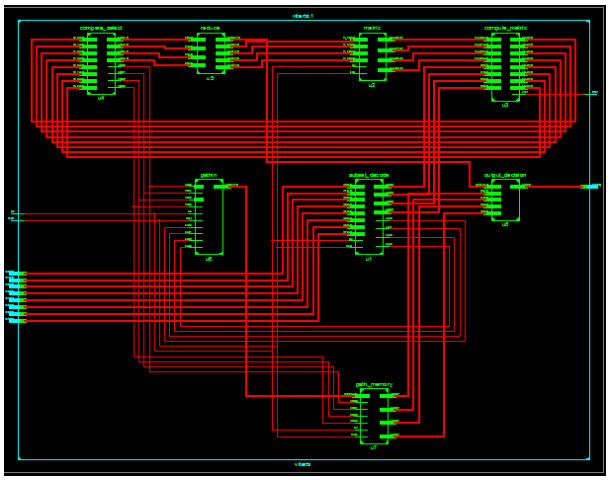


Figure 4.2b: RTL Schematic diagram of viterbi decoder

4.3 The Simulation Result of Viterbi Decoder:

The below diagram shows the result of viterbi decoder for the rate k=1/2 and 2/3.

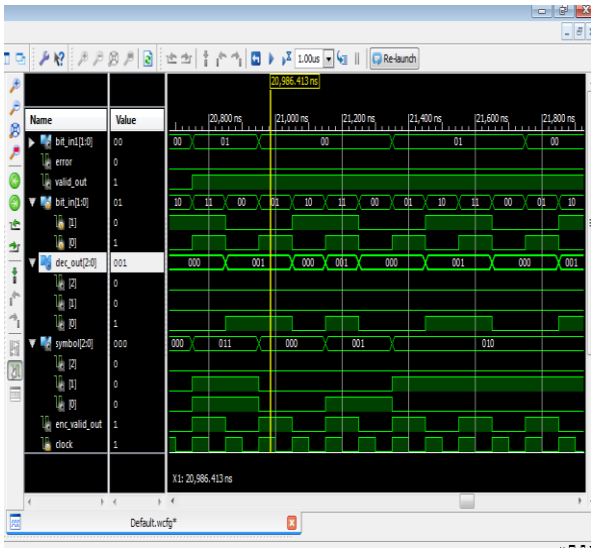


Figure 4.3a: Result of viterbi Decoder for the code rate k=1/2

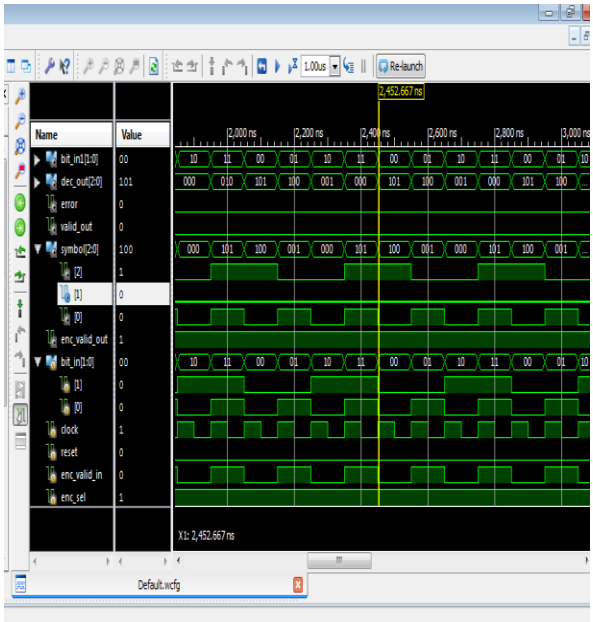


Figure 4.3b: Result of viterbi decoder for the code rate k=2/3

ISSN: 2278 – 1323

5.CONCLUSION:

In this paper we presented an implementation of the Viterbi Decoder with constraint length of 3 and variable code rate of 1/2 or 2/3 , The proposed solution has proven to be particularly efficient in terms of the required FPGA implementation resources so as Chip Silicon Area, Decoding Time and Power Consumption. We have implemented Viterbi Decoder on Spartan 3E FPGA by utilizing both method and Synthesis result shows that Trace back method is more efficient in term of Chip Area Utilization so as will be Power Consumption in comparison with Register Exchanged Methods..

6 FUTURE SCOPE:

This work can be further implemented to design and implement of a generic viterbi decoder. Some of the generic parameters are basic decoder specifications, metric size, trellis window length, number of surviving paths and pipeline depth

7.REFERENCES:

1. Inyup Kang, Member IEEE and Alan N. Wilson (1998). "Low Power Viterbi Decoder for CDMA Mobile Terminal", IEEE Journal of Solid State Circuits. IEEE. Vol 33. p.p. 473-481, 2010.
2. Viterbi, A."Convolutional codes and their performance in communication systems "IEEE Trans. Commun. Technol" ,VOl .Com 19, no ,5, Oct.1971, pp.715-772, 2009.
3. John G. Proakis (2001). "Digital Communication". McGraw Hill, Singapore. pp 502-507, 471-475, 2010.
4. Hema.S, Suresh Babu.V, Ramesh P, "FPGA Implementation of Viterbi Decoder", 6th WSEAS Int. Conf. on Electronics, February 2007.
5. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. Inform. Theory, vol. IT-13, pp. 260-269, Apr. 1967.
6. John G. Proakis (2001). "Digital Communication". McGraw Hill, Singapore. pp 502-507, 471-475.
7. Viterbi, A."Convolutional codes and their performance in communication systems "'IEEE Trans. Commun. Technol ,VOl .Com 19, no ,5, Oct.1971, pp.715-772.
8. S.Haykan, Communication Systems, Wiley, 1994.

8.AUTHOR BIOGRAPHY:

T.VENU GOPAL is from HYDERABAD, TELANGANA..In the Year 2005,Completed M.E in ECE with specialization **Communication systems** from **CBIT**, Gandipet affiliated by OU. In the Year 1992, Completed B.TECH in **ECE** from V.R.SIDDARTHA ENGINEERING COLLEGE. Currently working as an **Associate Professor** in ECE department, Vidya jyothi Institute of Technology, R.R Dist, Moinabad.He is having 20 years of teaching experience.His Areas of research interests include wireless communication, image processing mobile communication, signal processing, optical and satellite communication.

ISSN: 2278 – 1323