

## Research Article

### Pipelined Viterbi Decoder Using FPGA

Nayel Al-Zubi

Department of Computer Engineering, Al-Balqa Applied University, P.O. Box: 19117, Al-Salt, Jordan,  
Tel.: +962-777411142

**Abstract:** Convolutional encoding is used in almost all digital communication systems to get better gain in BER (Bit Error Rate) and all applications needs high throughput rate. The Viterbi algorithm is the solution in decoding process. The nonlinear and feedback nature of the Viterbi decoder makes its high speed implementation harder. One of promising approaches to get high throughput in the Viterbi decoder is to introduce a pipelining. This study applies a carry-save technique, which gets the advantage that the critical path in the ACS feedback becomes in one direction and get rid of carry ripple in the "Add" part of ACS unit. In this simulation and implementation show how this technique will improve the throughput of the Viterbi decoder. The design complexities for the bit-pipelined architecture are evaluated and demonstrated using Verilog HDL simulation. And a general algorithm in software that simulates a Viterbi Decoder was developed. Our research is concerned with implementation of the Viterbi Decoders for Field Programmable Gate Arrays (FPGA). Generally FPGA's are slower than custom integrated circuits but can be configured in the lab in few hours as compared to fabrication which takes few months. The design implemented using Verilog HDL and synthesized for Xilinx FPGA's.

**Keywords:** Convolutional encoding, FPGA, pipelined, Viterbi decoder

## INTRODUCTION

Convolutional codes are preferred over block codes due to simple decoding using the Viterbi algorithm with soft-decisions. The convolution coder is often used in many digital transmission systems (deep space communication, satellite communication, cellular and most wireless communication) where the signal to noise ratio is low. It is also used in storage devices as hard disks and compact disks to enhance retrieving data and lengthen the life time of the components (Lin and Costello, 1982).

The convolution coder achieves error free transmission by adding enough redundancy to the source symbols. The choice of the convolution code depends mostly on the application which is a matter of compromise between complexity, power, performance and bandwidth. In this design, an implementation code of a pipelined Viterbi decoder is achieved which supports a convolutional code based on a rate  $1/2$ , constraint length  $K = 7$  and sequence generating functions  $g_0 = 133_8$  and  $g_1 = 171_8$ .

This study will study the effect of the quantizer limits and the number of the soft-decision bits on the gain of the Viterbi decoder and the increase in speed of the Viterbi decoder using pipelining technique with complexity in hardware, all this carried out using MATLAB simulation and FPGA cards implementation.

The classical method to realize the Viterbi decoder is an iterative calculation and memory trace-back,

which is effective for a moderate decoding speed and long constrain length. For a high-speed decoder, iterative calculation and memory trace back becomes



Fig. 1: Viterbi decoder main units

the bottlenecks for the throughput which also depends on the word length. This study presents a pipelined method to realize the Viterbi decoder using an FPGA, in which the critical path is broken down saving time and increasing the speed of the decoder. Other effect on the Viterbi decoder which could increase the gain of the decoder and enhancing the Signal-to-Noise Ratio (SNR) which is affected by the number of the soft-decision bits and quantizer limits with all these the throughput can reach 4 times faster than the normal Viterbi decoder.

**Viterbi decoder architecture:** The Viterbi Decoder can be implemented with three basic units as in Fig. 1. The Branch Metrics Unit (BMU) calculates the branch metrics for each incoming bit. The Add-Compare Select Unit (ACSU) adds the branch metrics to the path metrics to calculate the new path metrics. Then it compares between the path metrics, the minimum or maximum, according to the implementation, to select the best path. The Survivor Memory Unit (SMU)

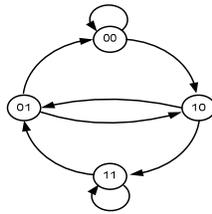


Fig. 2: State diagram

processes the outputs of the ACSU to produce the decoded bits.

The operational speed of the Viterbi decoder is limited by the ACS unit. To implement high speed Viterbi decoder, we can introduce pipelining. It is easily to do that in the BMU and SMU since they are purely feed forward paths. The problem holds in the ACSU due to the feedback path. In this unit the paths metrics are updated each time step, this cause the feedback. This can be seen from the state diagram in Fig. 2.

Taking a closure look to the state diagram where the ACSU is showing as in Fig. 3. High throughput-rate is achieved as we make the critical path very short. The critical path of a synchronous-circuit is defined as the path between two buffers (e.g., flip-flops) with the largest propagation delay and hence determines the maximum achievable clock frequency of the circuit. As shown in Fig. 4 the critical path here is in the feedback loop.

At Bit level, we can show the critical path in detail, assuming that the word length is 4-Bits. As shown in Fig. 4 (Fettweis and Meyr, 1991), the critical path

Table 1: Redundancy in CS-representation

$c_i$	$s_i$	$v_i$
0	0	0
0	1	1
1	0	1
1	1	2

extends for 4 adders and 4 maximum (or minimum) selections. This is for the carry ripple adders, where the carry should propagate from the LSB to the MSB. The compare decision should start from the MSB of the maximum (or minimum) selector down to the LSB hence, which depends on the word length.

**Carry-Save (CS) representation:** In carry save representation, the carry does not propagate to the next adder; instead it is saved with the sum of the next adder as shown in Fig. 5b. Figure 5a shows the carry ripple representation, here the second adder should wait for the carry of the first adder so the word length affects the speed of the adder, where in carry save mode the two adders takes the same time.

In CS the carry and the sum are combined to a new value,  $v_i = s_i + c_i$ , which can take on the values  $v_i \in \{0, 1, 2\}$  where the  $S = \sum_i (v_i) 2^i = \sum_i (s_i + c_i) 2^i$ .

Carry-Save representation has redundancy entries, this comes from either ( $c_i = 1$  and  $s_i = 0$ ) or ( $c_i = 0$  and  $s_i = 1$ ), where their sum equals to 1. This is shown in Table 1.

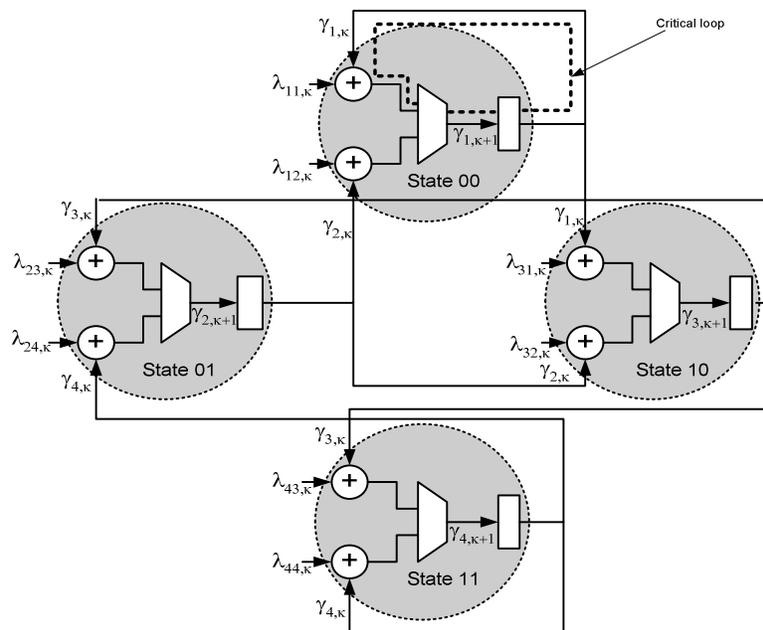


Fig. 3: A state diagram showing adders and maximum selection

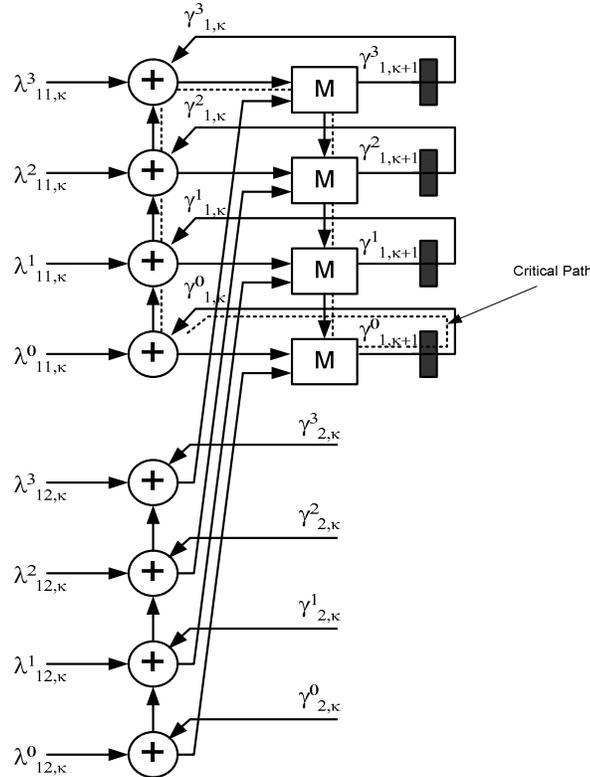


Fig. 4: State 00 of the state diagram in bit level with word length equal 4, in carry-ripple format

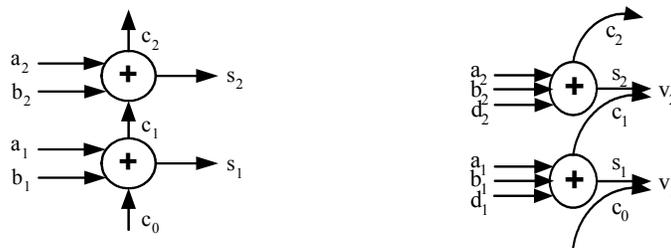


Fig. 5: (a) Carry ripple adder, (b) Carry save adder to add  $a = a_1 \cdot 2^0 + a_2 \cdot 2^1$  and  $b = b_1 \cdot 2^0 + b_2 \cdot 2^1$

**Carry-Save (CS) maximum selection:**

**CS comparison technique:** Redrawing Fig. 4 using carry-save maximum selection is shown in Fig. 6. The function of the CSM (Carry Save Maximum selection) block (Wicker, 1995; Ciletti, 1999) is to select the maximum of the inputs A or B, which are in carry save format. So the output is  $G = \max(A, B)$ , where  $A = \sum_{i=0}^{W-1} a_i 2^i$ ,  $B = \sum_{i=0}^{W-1} b_i 2^i$  and  $G = \sum_{i=0}^{W-1} g_i 2^i$ ,  $a_i, b_i, g_i \in \{0, 1, 2\}$ .

The carry save maximum selection starts at MSB and there is three cases for  $a_{MSB}$  and  $b_{MSB}$  where, MSB is  $w-1$  assuming the word length is  $w$ -bits. Figure 7 shows two bits of  $w-1$  and  $w-2$ .

**Case-1:  $a_{w-1} - b_{w-1} = 2$ :** In this case the minimum value of A is  $2^w$  this by setting all  $a_i = 0$  for  $i < w-1$ . And the maximum value of B by setting all  $b_i = 2$  for  $i < w-1$ , is  $\sum_{i=0}^{W-2} 2(2^i) = 2 \left[ \frac{1-2^{W-2+1}}{1-2} \right] = 2^W - 2$ . So A is always greater than B, then  $G = A$ .

**Case-2:  $a_{w-1} - b_{w-1} = 0$ :** Here  $a_{w-1} = b_{w-1}$  and  $g_{w-1} = a_{w-1}$  or  $b_{w-1}$ , so no decision made here till see the next stage. And no problem if  $g_{w-1}$  assigned to  $a_{w-1}$  or  $b_{w-1}$ .

**Case-3:  $a_{w-1} - b_{w-1} = 1$ :** This comes from either  $a_{w-1} = 1$ ,  $b_{w-1} = 0$  or  $a_{w-1} = 2$ ,  $b_{w-1} = 1$ . In this case the minimum value of A is  $2^{w-1}$  this by setting all  $a_i = 0$  for  $i < w-1$  and

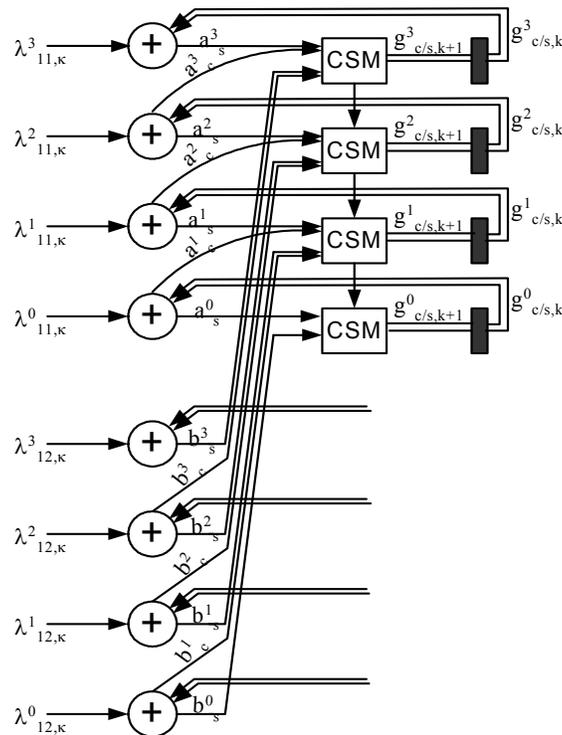


Fig. 6: State 00 of the state diagram in bit level with word length equal 4, in carry-save maximum format

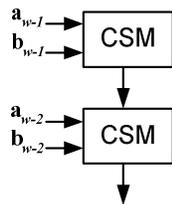


Fig. 7: CSM of  $a = a_{w-1} \cdot 2^{w-1} + a_{w-2} \cdot 2^{w-2}$  and  $b = b_{w-1} \cdot 2^{w-1} + b_{w-2} \cdot 2^{w-2}$

the maximum of B by setting all  $b_i = 2$  for  $i < w-1$ , is  $\sum_{i=0}^{w-2} 2(2^i) = 2 \left[ \frac{1-2^{w-2+1}}{1-2} \right] = 2^w - 2$ . The difference  $A_{\min} - B_{\max} = 2 \cdot 2^{w-1}$ . This case is called a *pre-decision for A*, since the difference can be greater than, less than or equal to, no final decision can be made on this bit level at this case; the decision can be made at the next level. Where there are three possible cases:

**Case-3.1:  $b_{w-2} = 0$ :** In this case the minimum of A is  $2^{w-1}$  and the maximum of B is  $2^{w-1}-2$ , the difference  $A_{\min} - B_{\max} = 2$ , so that A is the maximum and  $G = A$ .

**Case-3.2:  $a_{w-2} = 0, b_{w-2} = 1$ :** In this case the minimum of A is  $2^{w-1}$  and the maximum of B is  $2^{w-1}-2+2^{w-2}$ , the difference  $A_{\min} - B_{\max} = 2 \cdot 2^{w-2}$ , again no final decision can be made at this level as in case 3 above, we should look at the next level.

**Case-3.3:  $a_{w-1} = 0, b_{w-2} = 2$ :** In this case:

$$a_{w-1} + a_{w-2} = (1 \cdot 2^{w-1}) + (0 \cdot 2^{w-2}) = 2^{w-1}$$

and

$$b_{w-1} + b_{w-2} = (0 \cdot 2^{w-1}) + (2 \cdot 2^{w-2}) = 2^{w-1}$$

So to this level A equal B and the pre-decision of A is removed and the decision procedure starts on the next bit-level. But there is no error as we assign  $g_{w-1} = a_{w-1}$  and  $g_{w-2} = a_{w-2}$ , since there summation is equal.

These cases are shown in the flow chart for branch A in Fig. 8 and for branch B in Fig. 9. Where,  $(\delta i_A = \{c_A, s_A\} - \{c_B, s_B\})$  represents the difference  $a_i - b_i$ .

The design of the Carry-Save Maximum selection circuit was proposed in Gemmeke *et al.* (2002) and Gierenz *et al.* (2000) the design based on local comparison, which is for each branch at each state there is a CSM selection circuit that compare its metric with the maximum of the other branch metrics.

Figure 8 and 9 shows that the pre-decision ( $d_p$ ) flag determines the maximum bit  $a_i$  or  $b_i$  at the current bit. So by inhibiting the current bits of each branch with its pre-decision flag and O-ring the two inhibited branches we get the maximum branch bit at the current bit level, this illustrated as a function in (1) and (2):

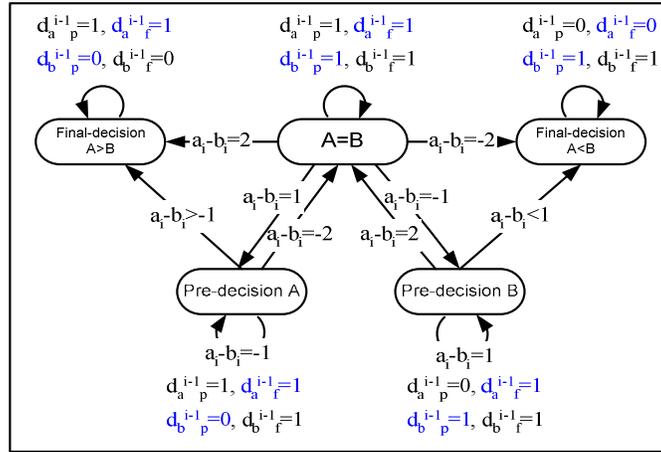


Fig. 8: The maximum selection flow chart, indicating the deference  $a_i - b_i$ , which is local for branch A

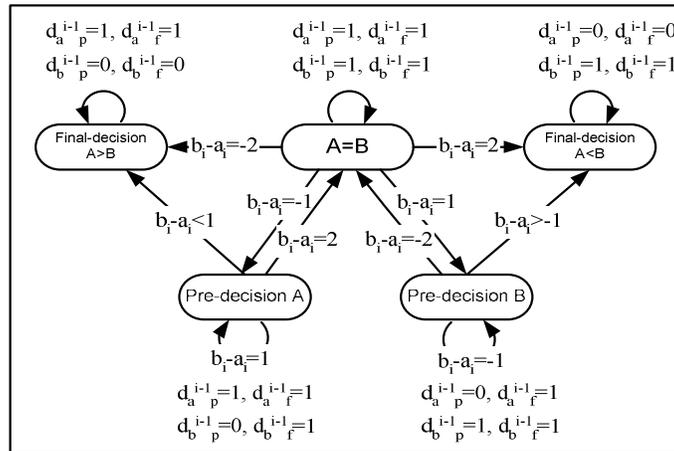


Fig. 9: The maximum selection flow chart, indicating the deference  $b_i - a_i$ , which is local for branch B

$$c_{max} = ((dp_a \cdot c_a) + (dp_b \cdot c_b)) \quad (1)$$

$$s_{max} = ((dp_a \cdot s_a) + (dp_b \cdot s_b)) \quad (2)$$

We modified this algorithm to work as minimum selection, this done by inverting the bits of each branch that entering the compare block and then inverting the output bits.

$$\sim c_{max} = ((dp_a \cdot \sim c_a) + (dp_b \cdot \sim c_b)) \quad (3)$$

$$\sim s_{max} = ((dp_a \cdot \sim s_a) + (dp_b \cdot \sim s_b)) \quad (4)$$

**CS redundancy suppression:** The first step in designing of the Carry-Save Maximum selection is the redundancy suppression circuit (Gemmeke *et al.*, 2002; Gierenz *et al.*, 2000) to eliminate the redundancy entry

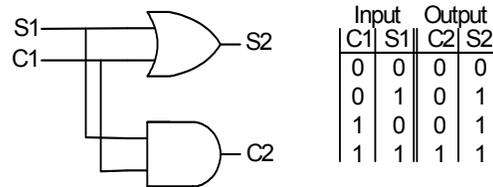


Fig. 10: Redundancy suppression circuit, with input and output table

in the Table 1 to simplify and reduce the complexity of the maximum selection circuit. This is shown In Fig. 10.

**CS truth table and decision flags:** Figure 8 shows the two required indicators for each branch. One indicates the pre-decision ( $d_p$ ) and the other indicates the final decision ( $d_f$ ). These flags should represent the five states in Fig. 8 and 9, this shown in Table 2.

Table 2: Indication of decision flags

$d_{a-f}^{i-1}$	$d_{a-p}^{i-1}$	$d_{b-f}^{i-1}$	$d_{b-p}^{i-1}$	Indication
1	1	0	0	Final decision A A>B
0	0	1	1	Final decision B A<B
1	1	1	1	A = B
1	1	1	0	Pre-decision of A (no final decision can be made)
1	0	1	1	Pre-decision of B (no final decision can be made)

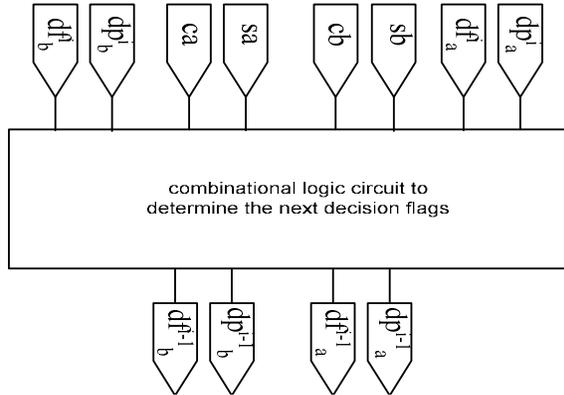


Fig. 11: Block diagram of current and next decision flags and current bits of branch A and B

To take into consideration the following bits of the word length, the difference of the following bits should

be inhibited by the decision flags. Table 3 shows the possible values of the input current decision flags ( $dp_a^i, df_a^i, dp_b^i, df_b^i$ ). The input difference of the current bits  $\delta i = \{ca, sa\} - \{cb, sb\}$  and the next output decision flags ( $dp_a^{i-1}, df_a^{i-1}, dp_b^{i-1}, df_b^{i-1}$ ) for the five states  $\{(A = B), (A>B), (A<B), (\text{pre-decision A}), (\text{pre-decision B})\}$ , Where Table 2 represents the truth table of Fig. 11.

**CS circuit implementation:** Logic minimization to Table 2 gives the following logic functions of  $dp_a^{i-1}, df_a^{i-1}, dp_b^{i-1}$  and  $df_b^{i-1}$  (2, 1-4), these functions and there circuit diagrams obtained by Electronic Workbench program which has the ability to do conversions between truth table, function and circuit implementation.

$$df_a^{i-1} = \{(dp_a^i(df_a^i(C'_a+S'_a))) + (dp_a^i(df_a^i(C'_aS'_a))) + (dp_a^i(dp_b^i(C'_bS'_b)))\}' + ((df_a^i(C'_aS'_a)) (dp_b^i(C'_bS'_b)))\}' + ((df_a^i(C'_a+S'_a)) (dp_b^i(C'_b+S'_b)))\}' \quad (5)$$

$$df_b^{i-1} = \{(dp_b^i(df_b^i(C'_b+S'_b))) + (dp_b^i(df_b^i(C'_bS'_b))) + (dp_b^i(dp_a^i(C'_aS'_a)))\}' + ((df_b^i(C'_bS'_b)) (dp_a^i(C'_aS'_a)))\}' + ((df_b^i(C'_b+S'_b)) (dp_a^i(C'_a+S'_a)))\}' \quad (6)$$

Table 3: Possible values of next decision flags ( $dp_a^{i-1}, df_a^{i-1}, dp_b^{i-1}$ , and  $df_b^{i-1}$ )

Inputs						Outputs							
$c_a$	$s_a$	$c_b$	$s_b$	$\delta i$	A?B	$dp_a^i$	$df_a^i$	$dp_b^i$	$df_b^i$	$dp_a^{i-1}$	$df_a^{i-1}$	$dp_b^{i-1}$	$df_b^{i-1}$
0	0	1	1	-2	A = B	1	1	1	1	0	0	1	1
0	0	0	1	-1		1	1	1	1	0	1	1	1
0	1	1	1	-1		1	1	1	1	0	1	1	1
0	0	0	0	0		1	1	1	1	1	1	1	1
0	1	0	1	0		1	1	1	1	1	1	1	1
1	1	1	1	0		1	1	1	1	1	1	1	1
0	1	0	0	+1		1	1	1	1	1	1	0	1
1	1	0	1	+1		1	1	1	1	1	1	0	1
1	1	0	0	+2		1	1	1	1	1	1	0	0
0	0	1	1	-2	Pre-decision A	1	1	0	1	1	1	1	1
0	0	0	1	-1		1	1	0	1	1	1	0	1
0	1	1	1	-1		1	1	0	1	1	1	0	1
0	0	0	0	0		1	1	0	1	1	1	0	0
0	1	0	1	0		1	1	0	1	1	1	0	0
1	1	1	1	0		1	1	0	1	1	1	0	0
0	1	0	0	+1		1	1	0	1	1	1	0	0
1	1	0	1	+1		1	1	0	1	1	1	0	0
1	1	0	0	+2		1	1	0	1	1	1	0	0
x	x	x	x	x	A>B	1	1	0	0	1	1	0	0
x	x	x	x	x	A<B	0	0	1	1	0	0	1	1
0	0	1	1	-2	Pre-decision B	0	1	1	1	0	0	1	1
0	0	0	1	-1		0	1	1	1	0	0	1	1
0	1	1	1	-1		0	1	1	1	0	0	1	1
0	0	0	0	0		0	1	1	1	0	0	1	1
0	1	0	1	0		0	1	1	1	0	0	1	1
1	1	1	1	0		0	1	1	1	0	0	1	1
0	1	0	0	+1		0	1	1	1	0	1	1	1
1	1	0	1	+1		0	1	1	1	0	1	1	1
1	1	0	0	+2		0	1	1	1	1	1	1	1

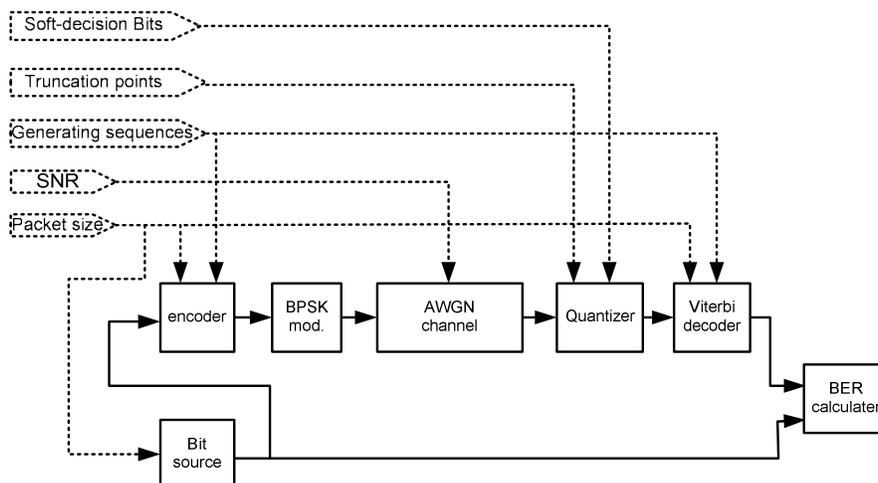


Fig. 12: Simple block diagram of the coding and decoding system

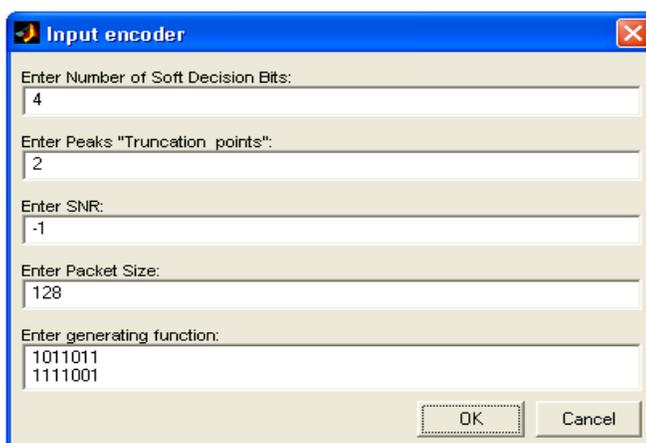


Fig. 13: Interface panel for MATLAB simulation program

$$dp^{i-1}_a = \{(dp^i_a(df^i_a(C'_a S'_a))) + (dp^i_a(df^i_a(C'_a + S'_a))(dp^i_b(C'_b S'_b)))' + (dp^i_a(df^i_a(C'_a + S'_a)))' + ((df^i_a((C'_a S'_a))) (dp^i_b(C'_b S'_b)))(dp^i_b(C'_b + S'_b))'\} \quad (7)$$

$$dp^{i-1}_b = \{(dp^i_b(df^i_b(C'_a S'_b))) + (dp^i_b(df^i_b(C'_b + S'_b))(dp^i_a(C'_a S'_a)))' + (dp^i_b(df^i_b(C'_b + S'_b)))' + ((df^i_b((C'_b S'_b)))(dp^i_a(C'_a S'_a))' (dp^i_a(C'_a + S'_a))'\} \quad (8)$$

**Viterbi decoder performance simulations:** The high level coding and decoding simulation of the Viterbi decoder was performed by MATLAB and Simulink. The built in Viterbi decoder of Simulink was used to verify our algorithm that was coded in MATLAB. Simple block diagram of the system is shown in Fig. 12. In the encoder simulation, the input data is

convolutional encoded according to the IEEE industry-standard 802.11a generator polynomials,  $g_0 = 133_8$  and  $g_1 = 171_8$  of rate  $R = 1/2$  and then modulated as BPSK. The channel model consists of an additive white Gaussian noise source to add noise to the modulated bits according to the SNR setup. The receiver demodulates the data to produce 4 bit quantized soft decision provided to the Viterbi decoder. A bit error rate comparator compares the decoded data with the source one to get the bit error rate. The interface panel for the MATLAB simulation program is shown in Fig. 13 the user can enter all parameters where this simulation environment can be modified to evaluate their effect on the Viterbi decoder performance.

The first entry box lets the user enters the number of soft decision bits that quantizes the received data to  $2^{\text{(soft-bits)}}$  levels. The second entry gives the choice of truncation of the received bits, that is, determination

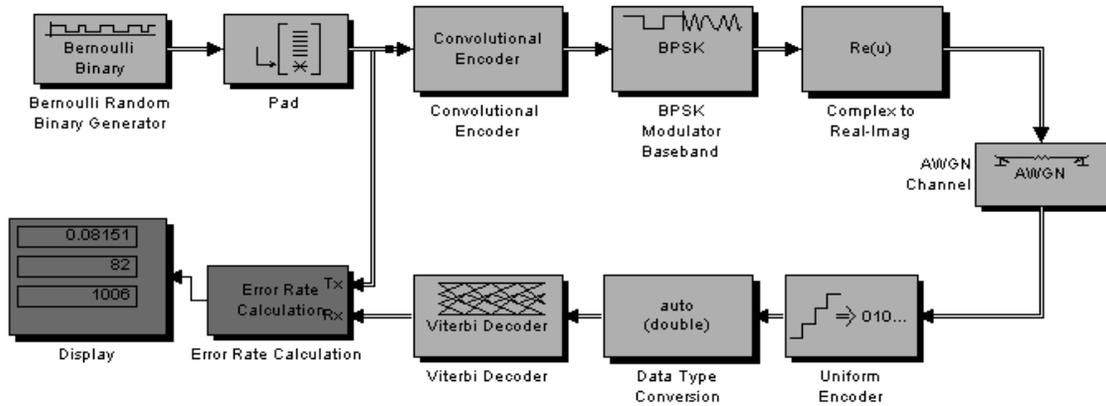


Fig. 14: Simulink block diagram

of limits of the quantizer, to be at the constellation points or beyond them. Signal-to-Noise ratio ( $SNR = E_b/N_o$ ) is entered in dB as a parameter in the third entry.  $SNR_{dB} = \log(E_b/N_o)$ , from this  $E_b = N_o * 10^{(SNR_{dB})}$  and for BPSK there is one bit per symbol so  $E_b = E_s$ . The packet size can be varied in the fourth entry, where the last entry box for the generated sequences to be entered.

The MATLAB code was written in two versions. One implements the carry-ripple Viterbi decoder and the other implements the pipelined carry-save were the Simulink simulate the Viterbi decoder in carry-ripple architecture. All architectures show exact simulation results for the same parameters.

The blocks of the Simulink are shown in Fig. 14. The first block generates packets of random binary data of fixed length using Bernoulli model. These bits feed a pad block that appends zeros to each packet to flush the encoder that is, ending at state zero for each frame. A convolutional encoder encodes the packet according to the given sequence functions. Then these bits are modulated as BPSK. At the receiver a quantizer with given number of soft bits and truncation point quantize the data. Viterbi decoder decodes the quantized data. Then bit error rate calculations are done on the decoded data and the source data.

Since the convolution code is standardized by the 802.11 committee, its performance should be acceptable to that of multi-path channels. Our verification consisted of exact performance of the bit level pipelined implementation to that of standard implementation. This is sufficient for testing using one channel model. We expect the performance of our implementation to match the standard for any channel type. Since hardware level simulations take considerably larger times, we did not perform multi-path simulations at hardware level.

**Coding gain vs. soft-decision bits:** The number of soft-decision bits affects the gain of the code. The

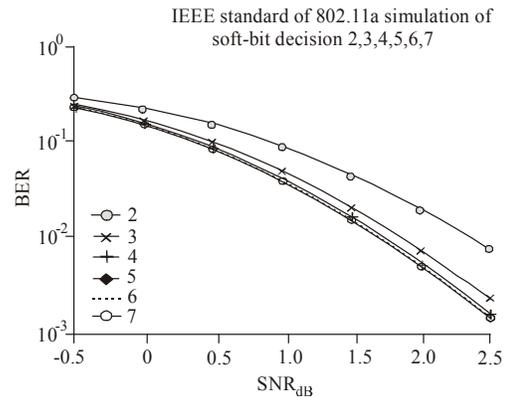


Fig. 15: BER vs. SNR for different soft-decision bits, for IEEE standard with code  $g_0 = 133_8$  and  $g_1 = 171_8$  of rate  $R = 1/2$ , with peak = 2

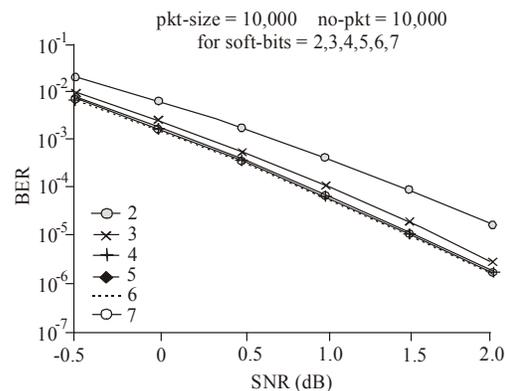


Fig. 16: BER vs. SNR for different soft-decision bits, for UMTS standard with code  $g_0 = 557_8$ ,  $g_1 = 663_8$  and  $g_2 = 711_8$  of rate  $R = 1/3$ , with peak = 2

simulation results are shown in Fig. 15. We see from the figure an increase in gain between 3-bits and 4-bits; however more than 4-bits gives very low increase in gain.

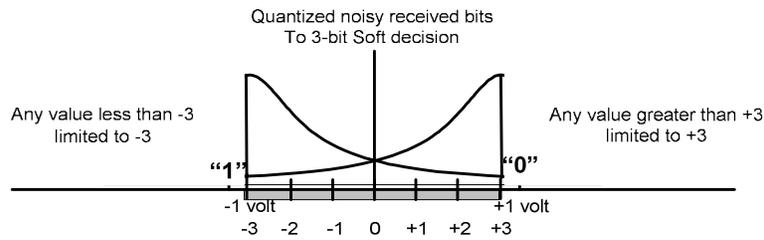


Fig. 17: Limits of the quantizer at the constellation points (+1, -1)

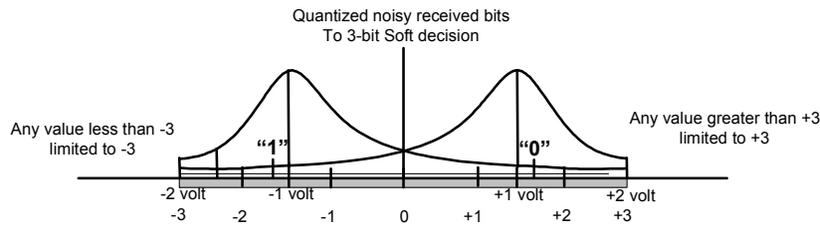


Fig. 18: Limits of the quantizer at (+2, -2) where the constellation points at (+1, -1)

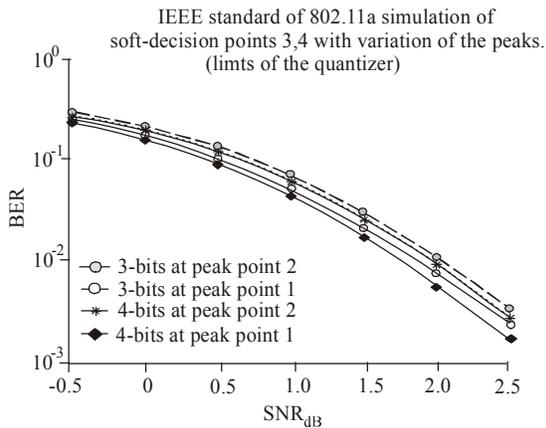


Fig. 19: Quantizer limits (+2, -2) and (+1, -1) effect on the code gain

Figure 16 shows the variation of the soft-decision bits on the error rate vs. signal-to-noise ratio of the UMTS convolutional encoding of generating sequences  $g_0 = 557_9$ ,  $g_1 = 663_9$ ,  $g_2 = 711_9$  of  $R = 1/3$  and constraint length = 9. It also shows good enhancement in BER for 4-bits soft-decision over the 3-bits. Increasing soft-decision bits more than 4-bits gives little gain as shown.

**Coding gain vs. peaks points (limits of the quantizer):** The limiting of the incoming data affects mostly the BER. This value is determined by the peak value. Figure 17 shows the quantizer with limits at the constellation points +1, -1, where Fig. 18 shows the limits of the quantizer at +2, -2 points, with constellation points at the +1, -1. For the two cases

the range is divided into the same number of levels are kept.

Figure 19 shows the effect of the quantizer limits on the 3-bits and 4-bits soft-decision bits. From the Fig. 19 we notice an approximately 0.5 dB gain for peak = 2 over peak = 1.

**Viterbi decoder circuit implementation:** The Viterbi decoder design was captured with Verilog HDL and simulated by Verilogger and Modelsim programs and synthesized by ISE Xilinx on a Virtex-II FPGA platform. A high level model was written in MATLAB and Simulink to carry out all the necessary simulation performance for different parameters.

## SYNTHESIS RESULTS

After the Viterbi decoder is verified by Simulink, it was synthesized by Xilinx (2000) ISE 5.1i development tool to prototype on an FPGA chip. We synthesized the two versions of Verilog HDL code that of carry-ripple and carry-save techniques. Figure 20 shows timing report of the two techniques; it shows the speed improvement of the pipelined carry-save technique over the carry ripple one, by a factor of 4.09.

Figure 21 shows the hardware complexity of the two versions before and after pipelining. Pipelining needs more hardware to accommodate the cut sets, redundancy suppression circuits and latency of the pipelining.

Many designs of the Viterbi decoder are implemented without taking the pipelining technique into consideration, while Santhi *et al.* (2008) designed a

<b>Synthesis speed before pipelining is</b>	
Timing Summary: -----	Speed Grade: -6
Minimum period: 34.987 ns (Maximum Frequency: 28.582 MHz)	
Minimum input arrival time before clock: 4.760 ns	
Maximum output required time after clock: 6.462 ns	
Maximum combinational path delay: No path found	
<b>Synthesis speed after pipelining is</b>	
Timing Summary: -----	Speed Grade: -6
Minimum period: 8.565 ns (Maximum Frequency: 116.754 MHz)	
Minimum input arrival time before clock: 6.552 ns	
Maximum output required time after clock: 6.462 ns	
Maximum combinational path delay: No path found	

Fig. 20: Timing report of viterbi decoder verilog HDL code of the two versions before and after pipelining

Hardware complexity before pipelining is	Hardware complexity after pipelining is
HDL synthesis report macro	HDL synthesis report macro statistics
Statistics	# Registers :973
# Registers :460	1-bit register :838
1-bit register :326	7-bit register :3
3-bit register :1	3-bit register :1
4-bit register :129	5-bit register :1
2-bit register :1	4-bit register :129
7-bit register :3	2-bit register :1
# Multiplexers :133	# Multiplexers :133
4-bit 128-to-1 multiplexer :1	4-bit 128-to-1 multiplexer :1
2-to-1 multiplexer :130	2-to-1 multiplexer :130
1-bit 4-to-1 multiplexer :2	1-bit 4-to-1 multiplexer :2
# Adders/subtractors :8	# Adders/subtractors :9
4-bit adder carry out :4	4-bit adder carry out :4
7-bit addsub :2	7-bit addsub :2
3-bit adder :1	3-bit adder :1
7-bit adder :1	7-bit adder :1
# Xors :200	5-bit adder :1
1-bit xor2 :200	# Xors :200
	1-bit xor2 :200
	# Counters :1
	5-bit up counter :1

Fig. 11: Hardware complexity report of Viterbi decoder Verilog HDL code of the two versions before and after pipelining

Viterbi decoder using pipelining and reached a 2.3 times faster than the decoder without pipelining.

### CONCLUSION

This study presents a comprehensive methodology for the design of a high-speed radix-2 64-state rate = 1/2 IEEE 802.11A/G Viterbi decoder, prototyped in an FPGA chip, where carry-save addition technique is used in developing the Viterbi decoder design. It was shown how efficiently this technique improves the speed of the Viterbi decoder approximately by a factor of 4, with more hardware complexity. In our high-level simulation we focus on the matching of the BER performance of this technique with ordinary carry-

ripple addition. We perform a soft-decision simulation, which shows a gain of 0.125 dB of 4-bit over 3-bit quantizer. And another simulation shows 0.25 dB gain if limits of the quantizer are beyond the constellation points.

### REFERENCES

- Ciletti, M., 1999, Modeling, Synthesis and Rapid Prototyping with Verilog HDL. 1st Edn., Prentice Hall, New Jersey.
- Fettweis, G. and H. Meyr, 1991. High-speed parallel viterbi decoding: Algorithm and VLSI-architecture. IEEE Commun. Mag., 29(8): 46-55.

- Gemmeke, T., M. Gansen and T. Noll, 2002. Implementation of scalable power and area efficient high-throughput viterbi decoders. *IEEE J. Solid-State Circ.*, 37(7): 941-948.
- Gierenz, V., O. Weib, T. Noll, I. Carew, J. Ashley and R. Karabed, 2000. A 550-Mb/s radix-4 bit-level pipelined 16-state 0.25  $\mu$ mCMOS Viterbi decoder. *Proceedings IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp: 195-201.
- Lin, S. and D. Costello, 1982. *Error Control Coding: Fundamentals and Applications*. 1st Edn., Prentice Hall, New Jersey.
- Santhi, M., G. Lakshminarayanan and S. Varadhan, 2008. FPGA based asynchronous pipelined viterbi decoder using two phase bundled-data protocol. *International SoC Design Conference*, pp: I314-I317.
- Wicker, S., 1995. *Error Control Systems for Digital Communication and Storage*. 1st Edn., Prentice Hall, New Jersey.
- Xilinx, Inc., 2000. *Virtex-II Platform FPGA Handbook*. Xilinx, San Jose, CA, pp: 560.