A Runtime Reconfigurable Architecture for Viterbi Decoding

Juan Manuel Campos, René Cumplido
Departamento de Ciencias Computacionales, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)
Luis Enrique Erro #1, Tonantzintla, Puebla, 72840, México
Phone +52 222 2663100 Ext. 8225 E-mail: {jcampos, rcumplido}@ccc.inaoep.mx

Abstract — **This paper presents the design and implementation of a runtime reconfigurable architecture for Viterbi decoding with a high throughput rate suitable for Software Defined Radio (SDR). SDR is a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software. The architecture can be reconfigured to decode convolutionally coded data with constraint lengths from 3 to 7 and code rates 1/2 and 1/3. Reconfiguration of the architecture does not require FPGA reprogramming. With a throughput of 70 Mbps, the proposed decoder is suitable for use in receiver architectures of 802.11a, 802.16, 3G and GSM.**

Keywords —**Reconfigurability, SDR, Viterbi Decoding**

I. Introduction

Forward Error Correction (FEC) schemes are an essential component of wireless communication systems. Typically convolutional codes are employed to implement FEC but the complexity of corresponding decoders increases exponentially according to constraint length. Present wireless standards such as the Third Generation (3G) systems, GSM, 802.11a and 802.16 utilize some configuration of convolutional coding. The Viterbi algorithm [1] is the most widely used technique for detecting and correcting errors in communication systems based in convolutional coding and is adequate for data reception [2].

With the rapid emergence of wireless communications networks there is a great demand and growing interest in building devices or systems that can operate on several wireless standards and gain benefit through reuse of hardware. This has motivated the design and implementation of a high speed reconfigurable architecture with Viterbi Decoding capability, as is proposed in this paper.

This work is the first stage of a concatenated system (Reed-Solomon - Convolutional Code). This system is a powerful combination for correcting errors. The convolutional code is used to clean up the channel for the Reed-Solomon code, which in turn corrects the burst errors emerging from the Viterbi decoder. One proposed application for this kind of system is the NASA/ESA standard coding scheme for deep space missions [3] and satellite communications.

II. Viterbi Decoding

Viterbi decoding is a technique for performing maximum likelihood sequence detection on data that has been convolutionally coded. If convolutional encoder parameters constraint length (*K*), Code rate (r) and Generator polynomials (G) of any communication system are known, then a decoding system can be realized.

The decoding problem is to determine the path with the minimum path metric (PM) through the trellis, with path metric being defined as the sum of the branch metrics along the path.

The complexity of the Viterbi algorithm resides in the computation of 2^K branch metrics (BM) for a constraint length *K* at each time stage. In the Trellis diagram, there are two paths entering each node with a branch metric value at any stage t (Fig. 1). Thus, state metric computation involves calculation of two branch metrics per state per node. Fig. 1 shows the characteristic trellis for K=3 and code rate=½.

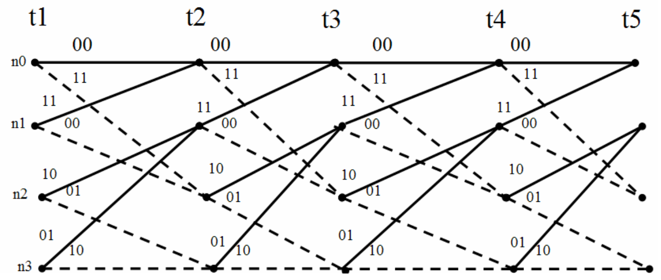

Figure 1: A typical Trellis

The Viterbi decoding algorithm is composed by the next stages:

a) Computing of Metrics
b) Add-Compare Select Operation
c) Trace back Operation

In the proposed design, the branch metrics are computed using the Hamming distance. The Hamming distance between binary data words *c*1 and *c*2, in this case denoted by *BM* (*c*1, *c2*) is the minimum number of bits that must be "flipped" to go from one word to the other [4].

IEEE Catalog Number: 06EX1386
ISBN: 1-4244-0403-7
Library of Congress: 2006925539

Fig. 1 shows a typical trellis, if the Hamming distance is used to calculate the branch metric between two states then for example:

$$BM(n0, n0) = BM(n1, n2)$$

$$BM(n1, n0) = BM(n0, n2)$$

Therefore it is only necessary calculate two BM per butterfly (one per node).

The next operation in Viterbi decoding is the Add-Compare-Select (ACS) which takes in two states metrics and two branch metrics and outputs the survivor path and an updated path metric at each node in the trellis and stores the updated path metrics into the path metrics register as input for the next stage of ACS.

The Survivor Management Unit (SMU) trace back through the trellis using the survivor paths to produce the output bits. Since the decoder generates decoded bits in inverse order, bit swapping is necessary by simply passing all the decoded bits through a LIFO memory.

The essence of the Viterbi algorithm resides in the operations ACS and trace back which need to be applied generally to a large number of nodes. This number of nodes is depending of constraint length (*K*).

III. RECONFIGURABLE VITERBI DECODER ARCHITECTURE

The functional block diagram of the reconfigurable Viterbi decoder architecture is shown in Fig. 2.

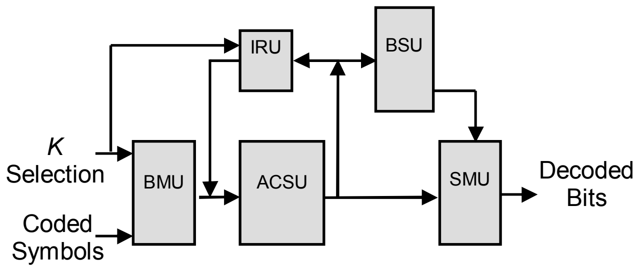

Figure 2: Viterbi Decoder

The architecture has two inputs, the constraint length and the data convolutionally coded and one output for the decoded bits. The output is a string of binary data.

Viterbi decoder is composed of five functional blocks:

a) Branch Metric Unit (BMU)
b) Initialize and Route Unit (IRU)
c) Add-Compare-Select Unit (ACSU)
d) Best State Unit (BSU)

a) Branch Metric Unit

The BMU is responsible for calculating 2^K branch metrics at each state of the trellis. Each branch metrics is computed as the Hamming distance between the received n-bit block and the actual codeword (ideal received n-bit block). The inherent symmetry in the trellis can be used to simplify the architecture of the BMU. This symmetry is used to reduce the number of path metrics calculations to one per node irrespective of the constraint length of the code.

The branch metrics are composed by 3 bits, therefore it is necessary calculate 8 different values of branch metrics and routing them to the correct ACSU. Although a small routing unit has to be added. This approach results in a smaller BMU. When the constraint length is defined, it is possible routing the correct branch metric to the corresponding processor.

b) Initialize and Route Unit (IRU)

The IRU initialize the architecture and provides the first path metrics to the ACSU also is the responsible for routing the path metrics depending of the constraint length *K*. This unit is composed by a ROM memory and multiplexes. The ROM memory contains 5 initialization vectors that are used according to the constraint length (*K*=3...7).

Each ACSU compares 2 path metrics according to (1)

$$n+1 \tag{1}$$

Where
n: Node bases to compare

E.g. If n=0 (this is the node 0) then the path metric of node 0 is compared with the path metric of node 0+1 (node 1) and the path metric of node 2 with the path metric of node 2+1 (node 3). In the architecture, routing operation is completely defined by (1)

c) Add-Compare-Select Unit (ACSU)

This unit executes the special purpose computation called Add-Compare-Select. The architecture of the ACSU is composed by 16 processors (Fig 3). Every processor computes 4 Butterfly or 8 nodes. Data are received from the IRU, processed (by the ACSU) and returned to the IRU.

A fully parallel scheme is selected to implement the ACSU. High throughputs are possible by using a fully parallel scheme.

The number of used processors depends on the constraint length K (2).

$$\text{Number\_of\_Processors\_used} = 2^{K-3} \quad (2)$$

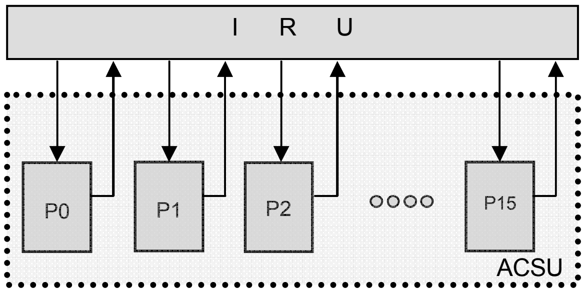

Figure 3: Add-Compare-Select-Unit (ACSU)

In turn, every processor includes 4 BC (Fig 4). Every BC takes in the path metrics from the IRU PM(n) and PM(n+1) The two candidates are compared and the smallest distance of the two path metrics is stored as the winning path metric, then every corresponding new path metric is calculated by adding the winning path metric and the corresponding branch metric.

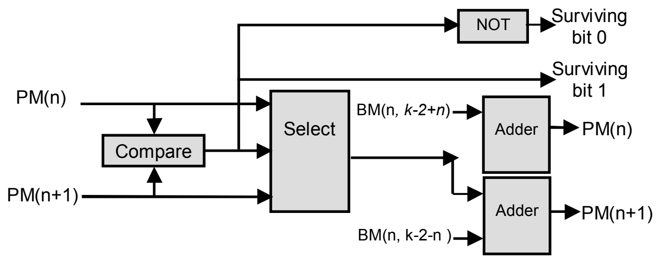

Figure 4: Basic Cell (BC)

Each surviving bit represents an individual node in the trellis diagram; therefore every BC processes a whole butterfly and generates two new path metrics. This novel configuration allows processing a whole butterfly with a minimum hardware.

d) Best State Unit (BSU)

Since the trace back operation is started from the best state of the Trellis, the BSU unit compares all the 2^{K-1} path metrics of the Trellis and selects the best state.

In the proposed architecture the best state is the state with the smallest path metric. This operation takes *K* clock cycles therefore this operation is begun *K* clock cycles before the beginning of the trace back operation.

e) Survivor Management Unit (SMU)

There are two hardware approaches for survivor management, namely register exchange and trace back [5]. We chose the trace back method over register exchange because the trace back is more suitable for reconfiguration purposes [6].

This unit finds the correct decoded data using the survivor bits generated by the ACSU. It consists of two survivor memory blocks, and a controller.

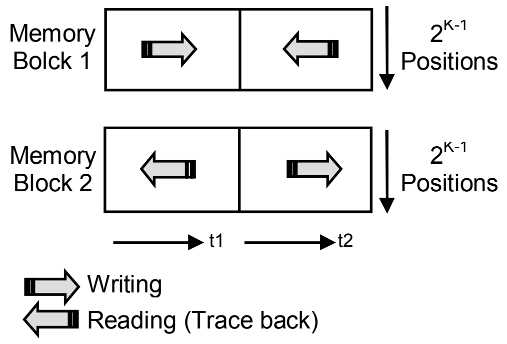

Figure 5: Survivor Memory Blocks

Two memory blocks are used (Fig 5). While memory block 1 is written, memory block 2 is read (t1). During the next stage (t2) memory block 2 is written when memory block 1 is read. This allows having a continuous flow of data out.

To write data in memory at the same time, data are concatenated in a single binary string and stored in a memory location, after this location is read and the data are separate to get the original information.

IV. RESULTS

The proposed reconfigurable architecture was modeled and simulated in Simulink V 6.0 R14 with Xilinx's System Generator. The target device to implementation is a Xilinx xc4vlx1000-10ff1513. The decoder uses 175k logic gates and 21kbits of memory for the trace back operation and the initialization of the architecture.

We verify that sequences of decoded bits are reconstructed in a continue flow after 25 cycles of initialization when reconfigurability options are established. Changing the constraint length (*K*) in runtime takes 2^K cycles. At a clock frequency of 70.02 MHz a throughout of 70.02 Mbps is achieved independently of the constraint length (*K*).

If the configuration of the decoder is changed to work with a scheme of packets instead of a streaming one, at a clock frequency of 70.02 MHz a throughout of 43 Mbps is achieved independently of the constraint length (K). Each sequence contains 32 decoded bits and it is produced every 57 cycles.

Several architectures for Viterbi decoder have been reported. Due to the characteristics of the presented architecture a comparison is possible with the next works.

In [6] a decoder (VITURBO) for constraint lengths 3 to 9 and code rates 1/2 and 1/3 is reported. A throughput rate of 60.5 Mbps is achieved with this implementation. Like the proposed architecture, this design realizes fully parallel scheme, where every trellis node is computed with a dedicated ACSU. Utilization of over 190k logic gates and about 327kbits of memory is reported.

In [7] an adaptive Viterbi decoding algorithm is used. Two implementations for constraint length K=4 to 9 and K=10 to 14 are reported. These implementations are based in a fully parallel scheme but very low throughput rates are achieved. Uses of memory resources are not mentioned.

In [8] a "Reconfigurable Viterbi for Mobile Platforms" for constraint lengths 7 and 9 and code rate 1/2 is reported. A throughput of 3.125 Mbps and 12.5 Mbps is achieved for K=9 an K=7 respectively. They report using 1314 logic elements and 35072 memory bits.

In [9] a reconfigurable Viterbi decoder architecture is presented for constraint length from 3 to 7. It uses a packet scheme achieving a throughput of 20 Mbps with an utilization of 89.5k logic gates.

TABLE 1
Comparison of reported architectures

	Constraint length supported	Throughput Max.	Area (Logic Gates)	Memory Resources (Kbits)
Proposed Architecture	3-7	70 Mbps	175k	21
[6]	3-9	60.5 Mbps	190k	327
[7]	4-9	333.7 Kbps	65k	-
[8]	7,9	12.5 Mbps	95k	35
[9]	3-7	20 Mbps	89.5k	-

TABLE 1 shows a comparison of key features with architectures previously reported. The architecture [6] presents high throughput but an excessive use of memory resources. The architecture in [7] presents a compact size but very low throughput. In [8] good management of memory resources is presented but low throughput is achieved. In [9] compact size is presented but low throughput is achieved in a packet scheme based.

V. CONCLUSIONS

In this paper a flexible runtime reconfigurable architecture for Viterbi decoder suitable for use in receiver architectures of 802.11a, 802.16, 3G and GSM was presented. The architecture is based in a fully parallel scheme and thus suitable for very high data rate decoding. The proposed architecture presents a high throughput of 70.02 Mbps and a low use of memory resources (21 Kbits).

The proposed architecture is easily scalable to other constraint lengths without diminishing the speed. The main contribution of this work is the novel and compact implementation of the Basic Cell to perform the Add-Compare-Select operation. This is possible thanks to the special branch metric calculation. The use of this compact ACSU allows having a small architecture and achieving high throughputs. Future work involves exploring techniques to reduce the power consumption, an important factor in mobile platforms. Power saving techniques ensures that the architecture is feasible for mobile devices.

REFERENCES

[1] J. G Proakis. "Digital Communications". McGraw-Hill, New York, NY, fourth ed., 2001.

[2] G. D. Forney, Jr. "The Viterbi Algorithm", Proceedings of The IEEE vol 61, pp. "268-278, Mar. 1978.

[3] S. Wicker, V. Bhargava. "Reed-Solomon Codes and their Applications" IEEE Press. 1994. ch 11.pp 242-260.

[4] A.Houghton. "The Engineer's Error Coding Handbook".Chapman & Hall, Great Britin, 1997, Ch 1, 3-6.

[5] R. Cypher and C.B. Chung. "Generalized Trace back techniques for Survivor Memory management in the Viterbi Algorithm" In GLOBECOM, Dec. 1990.

[6] J. Cavallaro, M. Vaya, "VITURBO: A Reconfigurable Architecture for Viterbi and Turbo Decoding", ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing 2003 – Proceedings 2, 497-500.

[7] S. Swaminathan, R. Tessier, D. Goeckel, W. Burlesson. "A Dinamically Reconfigurable Adaptive Viterbi Decoder". Proceedings of the 2002 ACM/SIGDA Thenth International Symposium on Field Programmable Gate Arrays , 227-236.

[8] R. Rasheed, A. Menouni, R. Pacalet. "Reconfigurable Viterbi Decoder for mobile platform" MWCN 2005, 7th IFIP International Conference on Mobile and Wireless Communications Networks, September, 19-21, 2005- Marrakech, Morocco.

[9] K. Chadha, J. Cavallaro, "A Reconfigurable Viterbi Decoder Architecture", Conference Record of the Asilomar Conference on Signals, Systems and Computers 2001. Vol.1, 66-71.