VLSI IMPLEMENTATION OF AN AREA–EFFICIENT ARCHITECTURE FOR THE VITERBI ALGORITHM

Carlos Cabrera

Department of Microelectronic
Catholic University of Peru
Lima. PERU

Montserrat Bóo and Javier D. Bruguera

Department of Electronic and Computers
University of Santiago de Compostela
Santiago de Compostela. SPAIN
elmboo@usc.es bruguera@gaes.usc.es

ABSTRACT

The Viterbi algorithm is widely used in communications and signal processing. Recently, several area–efficient architectures for this algorithm have been proposed. Area–efficient architectures trade speed for area by means of mapping the N states of the trellis describing the Viterbi algorithm to P processing elements, where $N > P$. In this paper a practical VLSI implementation of an area–efficient architecture to evaluate the Viterbi algorithm is presented. The architecture that has been implemented is composed of only two processing elements and the corresponding routing network to process, in different cycles, all the states of the trellis. The resulting architecture has been integrated in a chip using a 0.7μ CMOS technology, occupying an area of $9mm^2$.

1. INTRODUCTION

The Viterbi algorithm [6] is widely used in many decoding applications in communications and signal and image processing. It is known to be an efficient method for the realization of maximum likelihood decoding of convolutional codes. It is based on the study of a weighted graph, called *trellis*, which is used to reconstruct the actions of a convolutional encoder based on the information received through a noisy channel. Figure 1a) shows a trellis diagram of 16 states. The trellis describes the transitions between states and its scheme is repeated in time. The states of the trellis can be rearranged, as shown in figure 1b), to obtain a butterfly structure similar to that of the FFT. The maximum likelihood path through the trellis is calculated recursively by means of computing the optimum path of the N nodes of time t. The paths are represented by a

This work was supported in part by the Xunta de Galicia under contract XUGA20605B95. This work was performed while C. Cabrera was with the University of Santiago de Compostela

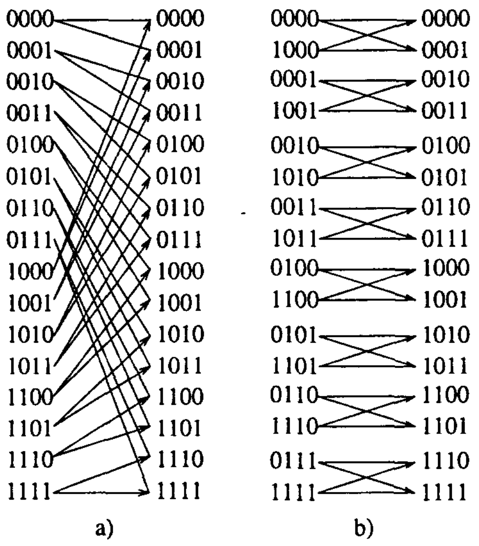

Figure 1: 16 states trellis diagram

path metric (PM). Each state receives the PM's of its preceding states and computes the optimum path,

$$PM[i]_t = \min_{all\ possible}(PM[k]_{t-1} + BM(k.i)) \quad (1)$$

where $PM[i]_t$ is the path metric of state i at time t and $BM(k,i)$ is the branch metric associated with the transition from state i to state k. The central unit of the Viterbi decoder is the calculation of the path metrics (equation (1)), a data–dependent feedback loop which performs an Add–Compare–Select (ACS) operation. This non–linear recursion is the only bottleneck for a high–speed parallel implementation

Most implementations for applications that require high speed processing, employ a state–parallel approach with one ACS devoted to each state of the trellis [3]. However, this approach is very expensive when the number of states is high. Alternatively, area efficient architectures [4] [7], where each ACS processes several

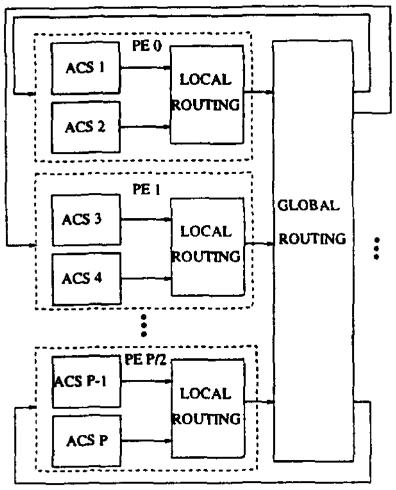

Figure 2: Architecture model.

states of the trellis in different cycles, can be used. In this kind of architectures the number of ACSs can be preset according to the speed and area requirements.

In this paper, we present the implementation, in an application specific integrated circuit, of an area efficient architecture composed of two processing elements (PE) for the processing of the Viterbi algorithm with an arbitrary number of states. The methodology for the mapping of the trellis onto a processor network of arbitrary size is described in [2]. Each PE is composed of two ACSs, in such a way that all the computations associated to a butterfly (two states) are processed in the same PE. Therefore, each PE computes several butterflies. On the other hand, the routing network, necessary to recirculate the states between the PEs, is also included. The resulting architecture has been integrated in a chip using a 0.7μ CMOS technology, occupying an area of $9mm^2$ and the maximum frecuency reached is $75MHz$.

2. AREA EFFICIENT ARCHITECTURE

The computation associated with a trellis involves three steps: a) Branch metric computation, b) PM updating in the ACS unit and c) Path storage and output sequence selection. The BM is computed as the Hamming distance from the values received from a noisy channel to the output produced in a noiseless channel.

The general structure of the area-efficient architecture is shown in figure 2. Each PE is made up of two ACSs which compute the two states corresponding to the same butterfly. This requires the use of the PMs of its two possible previous states. To do this there is a local routing in each one of the processors and a global routing between processors. This way, the data is recirculated and presented to each processor in the correct order and instant of time for processing the next state of the trellis.

The architecture implemented is based on the mapping methodology described in [2], where the global data flow of the Viterbi algorithm is described by means of three operators, concatenation, decimation and partial perfect unshuffle, which present a direct hardware projection. To this end, each state is represented by means of a three-dimensional index

$$[x,y,z] = [x_u,\dots,x_1,y_v,\dots,y_1,z_w,\dots,z_1] \qquad (2)$$

with $x_i, y_j, z_k = 0,1$. This way, the three-dimensional index is interpreted as a triad that indexes the PE where the state is processed, the cycle in which it is processed and the I/O path of the PE. That is, index $[x,y,z]$ can be interpreted as $[PE, CYCLE, PATH]$. Therefore, the global data flow defined as a perfect unshuffle

$$[x_u\dots x_1, y_v\dots y_1, z_w\dots z_1] \rightarrow \\ [z_1x_u\dots x_2, x_1y_v\dots y_2, y_1z_w\dots z_2]$$

is decomposed into three steps, described by the concatenation operator (β), the decimation operator (δ) and the partial perfect unshuffle operator (γ), defined as

$$\begin{aligned}\beta[x,y,z] &= [x_u\dots x_2, x_1y_v\dots y_1, z_w\dots z_1]\\ \delta[x,y,z] &= [x_u\dots x_1y_1, y_v\dots y_2, z_w\dots z_1]\\ \gamma[x,y,z] &= [z_1x_u\dots x_2, y_v\dots y_1, x_1z_w\dots z_2]\end{aligned}$$

That is, the concatenation operator introduces the last bit of field x in field y, the decimation operator introduces the last bit of field y in field x and the partial unshuffle operator performs a cyclic shift to the right over fields x and z. It can be shown that the sequential execution of these three operators results in a perfect unshuffle over the initial index. Note that the partial perfect unshuffle operator (γ) only modifies the PE and the $PATH$ fields, keeping unchanged the $CYCLE$ field. Therefore, it specifies the interconnection among PES. On the other hand, the other two operators modify also the $CYCLE$ field.

Table 1 shows an example of the application of the previous methodology for mapping a trellis with 16 states onto an architecture composed of two PE (four ACS units) with two I/O paths each. Therefore, the index of each state has three bits (8 states), with 1 PE bit (2 PEs), 1 CYCLE bit and 1 PATH bit (2 paths). Column labeled as $INPUT$ represents the inputs to the butterflies (one butterfly per PE) and columns labeled as β, δ and γ show the reordering of the data after the application of each operator.

PE	PATH	INPUT	β	δ	γ
	0	6 4 2 0		3 2 1 0	3 2 1 0
0			7 6 5 4 3 2 1 0		
	1	7 5 3 1		7 6 5 4	11 10 9 8
	0	14 12 10 8		11 10 9 8	7 6 5 4
1			15 14 13 12 11 10 9 8		
	1	15 13 11 9		15 14 13 12	15 14 13 12

Table 1: Application of the operators

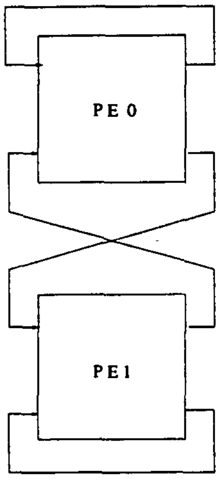

Figure 3: Interconnection between PEs.

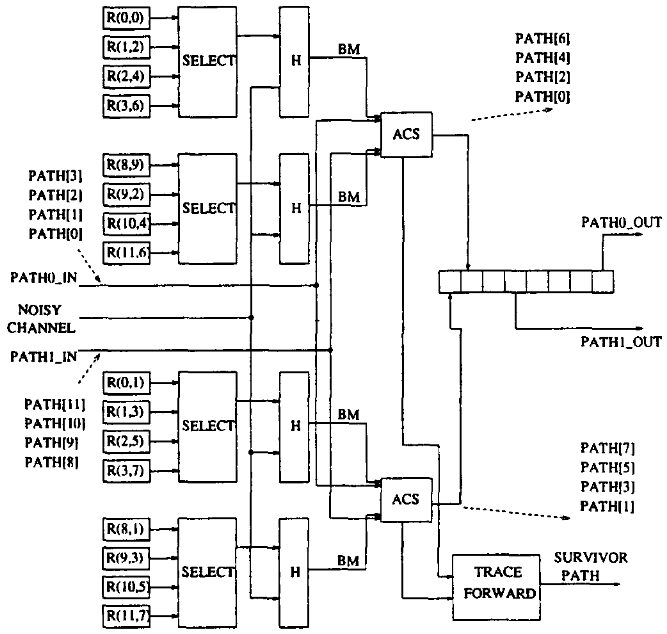

Figure 4: Internal structure of the PE

The concatenation and decimation operators are easily implemented by means of a FIFO queue with two input cells for concatenation and two output cells for decimation. On the other hand, the partial perfect unshuffle operator, as it represents the interconnection between PEs, is implemented as a routing network between processors.

3. IMPLEMENTATION OF THE ARCHITECTURE

As said before, we have implemented an architecture with two PEs, each of one is composed of two ACSs, to perform the Viterbi decoding of a 16 states trellis (see figure 1).

Figure 3 shows the interconnection between the two PEs. Note that it reflects exactly the operation of the partial unshuffle operator (see also table 1). The internal structure of the PE number 0 is depicted in figure 4. The structure of PE number 1 is similar. Each PE, that computes one butterfly, is composed of two ACS units, where the path metrics are updated, and a FIFO queue, with two input cells and two output cells, to implement the concatenation and decimation operators.

The input received through the noisy channel is processed in modules H to compute the Branch metrics (BM) as the Hamming distance from the value received to the value produced in noiseless channel (reference value). The reference values are stored in a set of registers R(i,j) representing the value that produces a transition from state i to state j. The path metrics (PM) of the states processed in the butterfly, received as inputs, together with the BMs, are used by the ACS units to obtain the new PMs. The PMs have to be recirculated to the following butterfly. The first part of the recirculation (concatenation and decimation operators) is performed in the FIFO queue.

The FIFO queue to implement the concatenation and decimation operators has, as input cells, the two first cells. In the first one (cell 0) the output of the first ACS is introduced, whereas, the output of the second ACS is introduce in the second cell (cell 1). After the computation and storage of the data associated with each butterfly, a two position right shift of the FIFO queue must be carried out to free the first two cells for the storage of the result of the computation of the next butterfly. Once the queue is full, that is, the outputs are obtained from cells 4 and 8, shifting one position the FIFO queue each time a data is read.

Moreover, each PE includes the hardware necessary to select the *survivor path*, that is, the PM through the trellis that matches the received sequence. Each processing cycle, the PE offers the decision bit that will permit reconstructing the state sequence that has occurred. The decision bits are in a RAM memory and the memory is traced back using the *trace forward* algorithm [1] in order to obtain the survivor path.

This architecture has been implemented in a integrated circuit using a $0.7\mu m$ CMOS standard cells (ES2) technology. The total area of the chip is $9mm^2$ and the maximum frecuency reached is $75MHz$. As two PEs (four ACSs) have been implemented to process trellises with 16 states, four butterflies of the trellis are processed sequentially in each PE. Considering that each butterfly is processed in one clock cycle, the total latency is four cycles.

It can be pointed out that, due to the recursive nature of the Viterbi algorithm, the PM's wordlength will grow with time. In order to avoid this, the PM has to be re-scaled after each iteration [5]. In our particular case, the wordlength need in the PMs is only four bits; that is, each ACS is composed of 4-bit adders and comparator. Therefore, carry ripple adders have been used.

The architecture implemented can be rearranged to perform the Viterbi decoding of trellises with a larger number of states. To do this, the logic to compute the BMs has to be adapted to the number of states.

4. CONCLUSIONS

A VLSI area-efficient architecture for Viterbi decoding of a convolutional code, represented by a 16 states trellis, has been described. The architecture is based in the mapping methodology proposed in [2]. That methodology allows to select the number of PEs needed to perform the decoding, in such a way that each PE processes several states of the trellis.

We have presented the implementation of an architecture composed by two PEs with two ACS unit each. This way, each PE processes four butterflies of the trellis. With few changes the architecture can process trellis with a larger number of states. Of course, in this later case, the latency will be larger.

The architecture has been implemented in $0.7\mu m$ CMOS technology, using the ES2 standard cells library. The maximum frecuency is $75MHz$. Therefore, the resulting implementation is suitable to be used in applications requiring high speed and/or low hardware cost.

5. REFERENCES

[1] P.J. Black and T.H.Y. Meng. *Hybrid Survivor Path Architectures for Viterbi Decoders.* Proc. ICASSP-93. pp. 433–436. (1993).

[2] M. Boo, F. Arguello, J.D. Bruguera, R. Doallo and E.L. Zapata. *High-Performance VLSI Architecture for the Viterbi Algorithm.* IEEE Trans. Communications (to appear).

[3] C-Y. Chang and K. Yao. *Systolic Array Processing of the Viterbi Algorithm.* IEEE Trans. Information Theory. Vol. 35. No. 1. pp. 76–86. (1989).

[4] F. Daneshgaran, K. Yao. *The Iterative Collapse Algorithm: A Novel Approach for the Design of Long Constraint Length Viterbi Decoders.* IEEE Trans. Communications. Vol. 43. No. 2/3/4. pp. 1409–1418. (1995).

[5] G. Fettweis and H. Meyr. *High Rate Viterbi processor: A Systolic Array Solution.* IEEE Journal on Selected Areas on Communications. Vol. 8. No. 8. pp. 1520–1533. (1990).

[6] G.D. Forney. *The Viterbi Algorithm.* Proc. of the IEEE. Vol. 61. No. 3. pp. 268–278. (1973).

[7] C.B. Shung, H.D. Lin, R. Cypher, P.H. Siegel and H. Thapar. *Area-Efficient Architectures for the Viterbi Algorithm-Part I.* IEEE Trans. Communications. Vol. 41. No. 4. pp. 636–644. (1993).