Design and Implementation of a Viterbi Decoder Using FPGAs

Bupesh Pandita
Analog Devices, Bangalore, India.
email: bupesh.pandita@analog.com

Subir K Roy
Indian Institute of Technology, Kanpur
Kanpur, India.
email: skroy@flab.fujitsu.co.jp

Abstract

This paper describes the design and implementation of Viterbi decoder using FPGAs. In this paper we explore an FPGA based implementation methodology for rapidly prototyping designs. We use high level synthesis to achieve this. Some of the implementation issues related to the Viterbi decoder, such as organization of path memory, decision memory reading techniques, and the clocking mechanism have been discussed.

1 Introduction

Viterbi decoder is an important block in any CDMA modem. CDMA systems being interference based use forward error correction schemes like convolutional encoding to increase cell capacity.The Viterbi Algorithm may be viewed as a solution to the problem of maximum a posteriori probability estimation of the state sequence of a finite-state discrete-time Markov process observed in memory less noise. A tutorial on Viterbi algorithm can be found in [1]. One of the main blocks of a modem used during forward-link demodulation is a Viterbi decoder. A normal Viterbi decoder for a constraint length of 9 (256 states) uses more than 15 kb of memory and can occupy as high as 1/3 of the modem chip area. Our aim was to design a 19.2 kbps, 256 state Viterbi decoder with the added capability of catering to higher input data rates. To the best of our knowledge none of the literature discusses Viterbi decoder implementation based on high level synthesis targeted for Field Programmable Gate Arrays (FPGAs). This has been the focus in the present paper.

Several important design issues, such as, organization of a path memory, decision memory, the decision memory reading techniques, and the clocking mechanism have not been made available due to the proprietary nature of the system implementation. For example, some of the implementations use twos' complement representation for the path metrics. However, this can only be achieved at the cost of an important metric used for input bit sequence synchronization. We aimed at retaining this bit-synchronization metric, even though it made the normalization of the path metrics essential.

Broadly VLSI architecture implementations of the Viterbi algorithm can be divided into two categories

- node parallel
- node serial.

In the node parallel approach, as the computations for all nodes in the trellis are done concurrently, the decoding speed is high. In the node serial approach, these computations are done sequentially by a single node and the decoding operation is much slower. These nodes or ACS (Add Compare Select) units decide which of the possible information sequences entering a state is more likely to survive. The ACS units need to be interconnected according to the trellis diagram; this gives rise to a problem of different kind where reduction of the wiring area on silicon chips is the main aim. To reduce global wiring which may occupy as high as 37% of the Viterbi decoder area, architectures reducing global wiring area and also with fewer ACS units have been proposed [2].

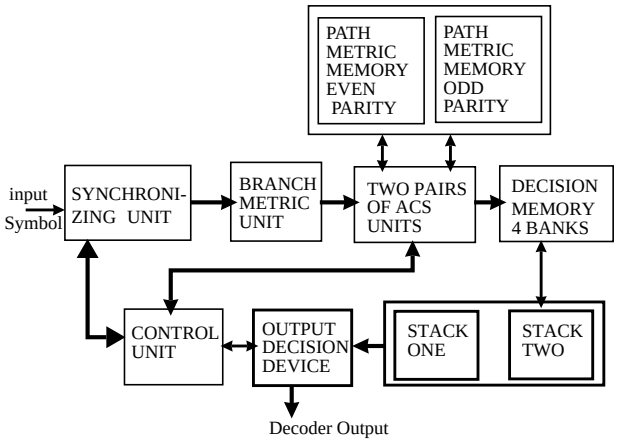

Figure 1 Viterbi decoder block diagram

In our Viterbi decoder implementation we have used two ACS pairs to process the data serially. Using a single ACS unit was found to increase the interconnections and also the benefit of the butterfly structure of the Viterbi algorithm is lost. Decisions from the ACS units are stored in a partitioned internal path memory. A trace back through the path memory of depth 64 outputs a single bit data.The path memory has been partitioned into even and odd parity memories, and further subdivided into four blocks. The main drawback of this configuration is the complicated memory decoding structure. However, this is

more than offset by the reduction of other interconnection resources and the increase in the resources utilization. Figure 1 shows the top level block diagram of the Viterbi decoder. In the decoder the 8-level quantized data is decoded optimally by the Viterbi algorithm. Not more than eight-levels of soft decision are used, as eight-levels of quantization results in a loss of less than 0.25 dB.

In this paper we present an implementation methodology which enables rapid prototyping. We chose FPGAs because of their reprogrammability and short programmability times. With FPGAs it is possible to immediately assess the impact of important architectural and design considerations on the final implementation [3].

2 Design Flow

We have used Verilog, a high level hardware description language for the Register Transfer Level (RTL) description of the Viterbi decoder. Data generators and pseudo-noise generators written in Verilog are used to specify the inputs during the simulation of the architecture. The in-built functions of Verilog are used to check the coding gain and hence the suitability of the architecture.One of the advantages of writing the RTL description in Verilog was the in-built random number generators which return integer values distributed according to the various probabilistic distributions e.g., poisson_distribu-tion (), Uniform_distribution () [4]. These functions were used not only to generate the input data stream while testing the proposed architecture, but also to simulate the noise bursts that affect the real time data transmission.To create an FPGA implementation, the initial Verilog file was modified to be acceptable to our Verilog FPGA interfacing tool- PINNAUQ [5]. PINNAUQ targets the RTL synthesis to the Xilinx FPGA architecture. The Verilog code was compiled together with the Xilinx technology libraries and the result was saved in the Xilinx netlist format (XNF) file format. XACT, the Xilinx developmental system is then run on the XNF file to create a configuration file for the FPGA. The performance of the synthesized design is obtained and this information is then used to hand tune the Verilog RTL code of the Viterbi decoder so as to effect the desired change through PINNAUQ.

3 Decoder Implementation

3.1 Specifications

The specifications chosen for the Viterbi decoder implemented are

1. K (constraint length)= 9 (256 states), R = 1/2 convolutional decoder,
2. Generator polynomials g(1) = 561 and g(2) = 743 , ((coding gain) = 4.77,
3. 8- level soft decision inputs,
4. Survivor path length 64.

Additionally it was decided that the decoder should have the added capability of self-synchronization in case the synchronization is lost. A microprocessor interfaced to the decoder reads the path metrics before initiating any corrective measures. The rate of growth of path metrics plus the rate of normalization was used as an indicator of decoder synchronization. The normalization circuit used consists of two counters, one clocked by data outputted and the next clocked by normalizing circuit. The difference of the counts of the counters gives the normalizing rate.

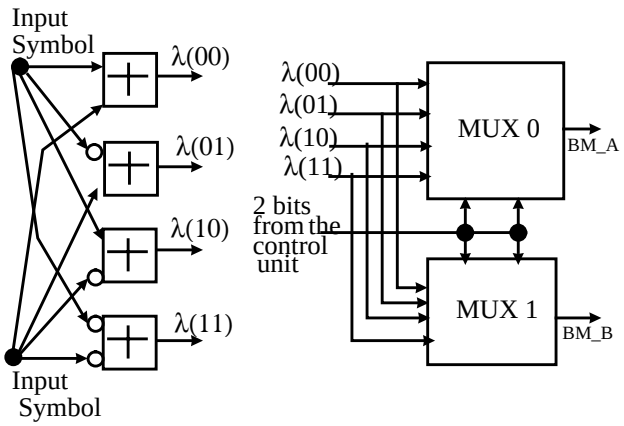

Figure 2 Branch metric unit block diagram

3.2 Branch Metric Unit

The two 3-bit soft decisions are combined to form four possible 4 bit branch metrics i.e., λ(0,0), λ(0,1),λ(1,0), λ(1,1). The branch metrics are generated as the Hamming distances of the input data stream from (00), (01), (10), (11); for example Hamming distance of two 3-bit soft decisions `101' and `000' from symbol `10' is 2 and from the symbol `01' is 12. The maximum value the Hamming distance can take is 14, and that explains using four bits to represent the branch metrics. The block diagram of the branch metric unit has been shown in Figure 2 . We found the gains accrued by using complicated metrics were minimal for survivor path memory more than 6 constraint lengths deep. While the path metrics are being generated, the branch metric unit is used to generate the branch metrics of the two new 3- bit data stream. The four-bit path metrics are stored in four 7-bit registers.

3.3 Add Compare Select Unit

In Figure 3 , area efficient 2- way ACS units have been designed using serial one-bit adders. The two comparators used to compare outputs of the two competing nodes have been implemented using serial one-bit subtracters, which ideally suits our requirement as it

results in a pipelined structure. The outputs of the subtracters are latched into two 7-bit registers. The MSB of these registers are sent to the decision memory. They are used to choose between the two competing paths. The adder outputs are held in four 7- bit registers. The MSB of the surviving path metric is used to find whether normalization is required or not. The path metrics are stored in the path metric memory with each location of length 7-bits. A control unit register holds the values of the smallest and the largest path metric in two registers, which are accessible to an external microprocessor. The normalization is initiated only during the next cycle.

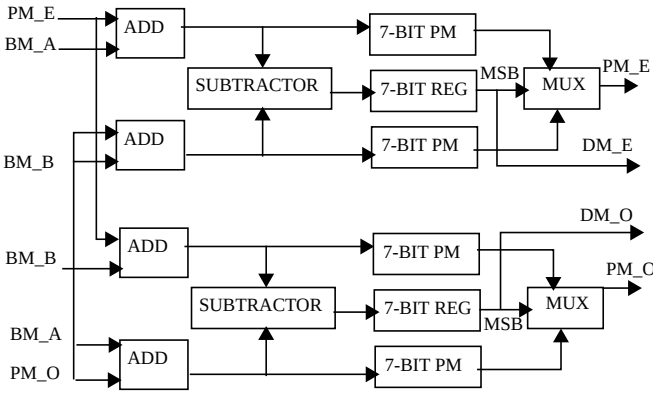

Figure 3 ACS units block diagram

3.4 Path Metric Memory

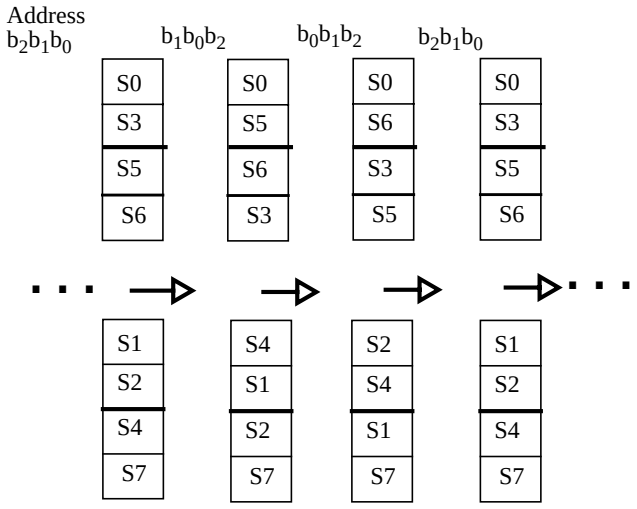

Figure 4 Address generation for in-place computation of path metric memory contents. Path metric memory has been divided into even and odd parity blocks and each block has been further subdivided into two blocks.

To accrue benefits from the butterfly structure of the Viterbi algorithm, the path metric memory has been divided into two blocks, i.e., even parity block and the odd parity block of sizes 128 x 7 each. We found the path metric could further be subdivided to allow more than two accesses at the same time. This can be useful where memory access is the main bottleneck. This allows parallel access of the two-path metrics required for the computation of the next path metric. Figure 4 shows addressing of the path metric memory. Address scrambling was used to allow in-place computation of the path metrics. This has obviated any need for a dummy buffer. We found the even and odd blocks could further be subdivided in order to speed the memory fetches [6].

3.5 Decision Memory

The traceback memory has been organized as a two dimensional structure, both the rows and the columns can be accessed. Figure 5 shows the organization of the decision memory. The number of the rows is equal to 2^7. Each column corresponds to one stage in the trellis.The total memory has been divided into 4 banks with each bank of size 2^7 x 32 dibits (decision memory may be viewed as a three dimensional structure). A dibit is a group of two bits. This feature is highly suitable for FPGA based implementations as well. This feature allows to use the same address for writing the two decision outputs from the ACS unit with a single address.

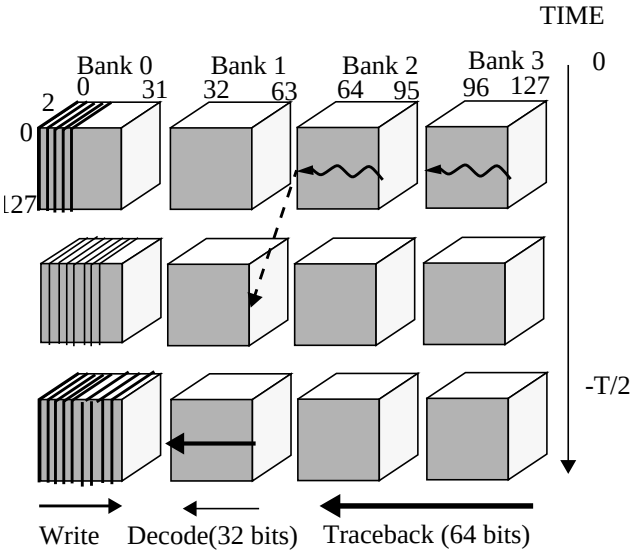

Figure 5 Decision memory structure

Power consumption is reduced at the architectural level by beginning each cycle with the block address decoding and then enabling the two blocks (one for reading and the second for writing). This sub-array architecture not only results in a faster implementation, but also consumes less power, since the number of banks active at a time is limited to two and hence less

capacitance is switched per read/write cycle. The only drawback is the area penalty due to the increased overhead of the decode logic, and their routing interconnections.

The data is decoded in bursts. The data is available after a trace back through two banks; and during the trace back through the third bank; the data is decoded at the rate of 3 bits per input symbol. The decoded data is written into a two-stack structure, which performs both the bit reversal as well as the elimination of the bursts. The first decoded symbol appears after the first 128 bits have been processed; thus, making the output latency equal to 128. A single stack structure can be used if read and write cycles are interleaved, or a dual port memory is employed for the stack. However, it complicates the circuit design.

4 Clocking Mechanism

A single-phase clocking scheme was preferred because it results in a simpler design style with reduced routing overheads. Clock signals, which drive each module are obtained from a common source and routed inside the chip to these modules.

5 Rapid Prototyping of the Viterbi Decoder with Xlinx FPGA

The Viterbi decoder was implemented using the Xilinx XC4000 FPGA series [7]. FPGAs allow shorter design turnarounds and reduced verification times; which can potentially result in large design savings. The generation of XNF description of our design was done using PINNAUQ. Once the proper XNF description has been generated, the mapping to the XC4000 series is straightforward using the Xilinx-supplied tools for file format conversion. The design is translated to a logic cell array (LCA) file from where a serial bit-stream file for configuring the CLBs is generated. Choosing the right implementation style for FPGA design can cause a huge difference in resource utilization. Table 1 shows the resource utilization for various descriptions of a adder. Xilinx provides a partial solution to this problem by supplying a library of adders, subtracters, counters, and comparators. We used these libraries in PINNAUQ to

Table 1: Resource utilization for different styles

Description	FMAPS	HMAPS	IOBS	Other Resources
ADD(XOR)	2	0	6	0
ADD(+)	1	0	6	0

implement complex functional units. PINNAUQ uses X-BLOX library functional units by including references to the X-BLOX modules in the Verilog code. During the processing by XACT, X-BLOX is invoked as module generator to synthesize appropriate functional units. Figure 6 shows how a X-BLOX description has been include in the Verilog description of one-bit adder. We found that the X-BLOX libraries generally helps in optimizing some modules; however, most of the modules cannot be optimized using these features. A better solution is to generate one's own circuits with XACT and to include them as components in Verilog description. This feature is readily supported by PINNAUQ; and we used it extensively to optimize our implementation.

```
Module one_bit_adder(a,b,reset,clock,sum);
 input [0:0]  a,b,reset,clock;
 output[0:0]  sum;
 reg   [0:0]  carry_in;
 wire  [0:0]  sum;
 wire  [0:0]  carry_out;
ADD_SUB_1_UBIN    adder(.A(a),    .B(b),    .
C_IN(carry_in),.FUNC(sum),.ADDER_SUB(1'b1),.
 C_OUT(carry_out));
  always @(posedge reset or posedge clock)
  if(reset == 1'b0) carry_in = 1'b0;
  else if(clock == 1'b1) carry_in = carry_out;
  endmodule
```

Figure 6 Including X-BLOX instantiation in a Verilog file.

6 Conclusions

Viterbi decoder is one of the most important blocks in a CDMA modem. In this paper we have designed and implemented the Viterbi decoder targeting a FPGA implementation. Our aim was to develop an area efficient 19.2kbps, 256 states Viterbi decoder, but the design can cater to higher input data rates [8].

References

[1] G.David Forney,Jr., "The Viterbi algorithm," *Proc. IEEE* , vol-61, Mar 1973.

[2] Kang et al., "Low-Power Viterbi decoder for CDMA mobile terminals", *IEEE J. Solid-State Circuits*, vol. 33 , Mar 1998.

[3] D.J.Coggins et al., "A comparison of path memory techniques for VLSI Viterbi decoders," *VLSI 89* pp.379-388.

[4] Cadence, Verilog-XL reference.

[5] SAS, PINNAUQ a FPGA implementation tool.

[6] Chandrakshan et al., "Low-Power chipset for portable multimedia I/O terminal," *IEEE J. Solid-State Circuits*. Vol.29, no.12, Dec 1994.

[7] Xilinx X-BLOX manual.

[8] Bupesh Pandita, "Design and Implementation of a Viterbi Decoder using FPGAs". *M.Tech thesis*, Department of Electrical Engineering,, IIT Kanpur, 1998.