FPGA Implementation of the Viterbi Decoder

University of California, Berkeley
CS-252 Graduate Computer Architecture Project Report
Iakovos Mavroidis
e-mail: iakovos@cs.berkeley.edu

December 17, 1999

Contents

Abstract

This report presents the design and implementation of the Viterbi decoder on FPGA. The goal of the project is to study area-speed-effectiveness trade-offs while in parallel propose a cost-effective implementation of the decoder. Several Viterbi decoders were built in order to compare their characteristics. The work of this project is avalaible at http://www.cs.berkeley.edu/˜ iakovos/cs252/viterbi.

1 Introduction

Need for reliable data transfer is becoming more and more important in today's digital communications. The Viterbi algorithm is widely used for the elimination of the potential noise in a data stream. It belongs to a large class of error correcting codes known as convolution codes. The decoder operates by finding the maximum likelihood decoding sequence. The usefullness of the Viterbi decoder is depicted in Fig. 1. At the source the Viterbi encoder encodes the input stream and transmits it to the destination via a noisy medium. The encoding is such that the Viterbi decoder can remove potential noise in the incoming stream by decoding it. The characteristics of the decoder are its effectiveness in noise elimination, speed of decoding and cost (hardware utilization).

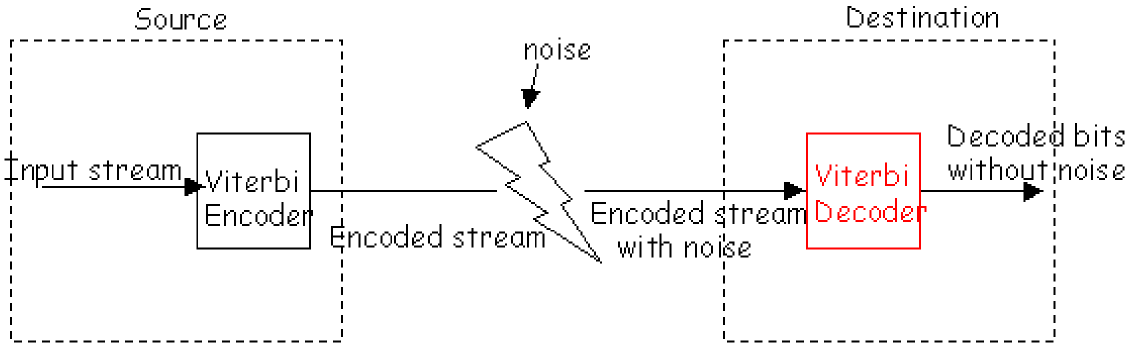

Figure 1: Use of the Viterbi Decoder

This report presents the design and implementation of the Viterbi decoder. Its streamed input-output, regular architecture and parallel execution favor an on-FPGA implementation. An extensive study on cost-speed-effectiveness trade-offs presented in this report can help to find the optimal cost effective Viterbi decoder. Ideally we should be able to trade-off any of the three components for any other. In order to better study these trade-offs we implemented a lot of Viterbi decoders using scripts to automatically generate their harware descriptions. This automation process was greatly facilitated by the regularity of the designs.

2 The Viterbi Decoder

A short description of the Viterbi algorithm is given in this section. The reader is referred to [1] for an in-depth description. A small system is chosen to illustrate the Viterbi encoding and decoding process. The algorithm used by the Viterbi Decoder belongs to a class of algorithms known as convolution codes. The rate of the convolution coder is defined as the number of input bits to the output bits. The rate of this encoding is 1/2, i.e. the system encodes 1 input bit to 2 output bits. The encoder is depicted in Fig. 2.

In order to encode bit x(n) from the input stream this encoder creates two bits, namely G0(n) and G1(n) using the last 3 input bits, i.e. x(n-1), x(n-2), and x(n). The encoding bits are produced from the equations :

- G0(n) = x(n) + x(n-1) + x(n-2) % 2
- G1(n) = x(n) + x(n-2) % 2

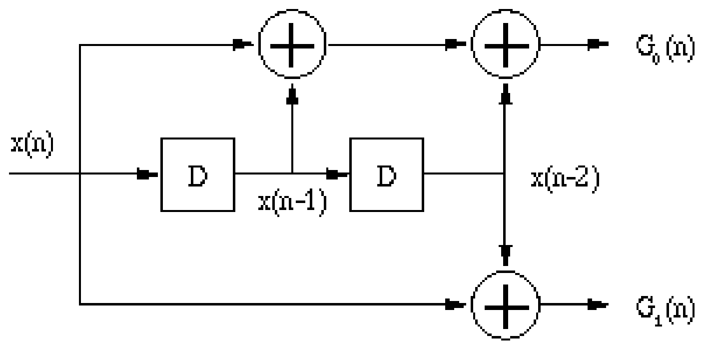

Figure 2: Block diagram of the Viterbi Encoder

The number of bits used for encoding one bit is called *"constraint length"* (in this case the constraint length is 3), and the equations that describe the encoding are called *"generator polynomials"*. At the destination the Viterbi decoder decodes the encoded stream providing the more likelihood decoding sequence. The *trellis diagram* depicted in Fig. 3 is used for the decoding. Each node in the trellis diagram denotes one of the four potential pairs (x(n-1), x(n-2)) of the last two decoded bits. The trellis diagram can be seen as a flow-control diagram where each node represents a state and transitions happen depending on the input stream. From any node we can make a transition to one of two other nodes corresponding to receiving a 0 or a 1 as bit x(n) at the input. The way the trellis diagram is constructed depends on the constraint length but not on the generator polynomials. The two numbers shown on every transition in the figure are the results of the above two generator polynomials.

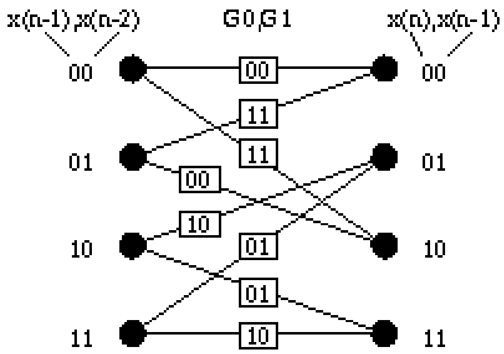

Figure 3: Two stages of the trellis diagram

Each stage, comprised by the four nodes at the same vertical position, in the trellis diagram is associated with one decoded bit and the corresponding two encoded bits. The Viterbi decoder finds the maximum likelihood path through the trellis diagram. Ideally the encoded bits should match with the numbers on the transitions of the resulting path. A transition also represents a 2-way communication of information between its two nodes. Each node is an *Add Compare Select* (ACS) module used during Viterbi decoding in order to perform its two major operations, *metric update* and *bakctrack* (or *traceback*). These operations are described in short in the Appendix.

2.1 Step-by-step operation of the Viterbi Decoder

The Viterbi Decoder consists of many ACS blocks. Each one includes two 8-bit adders and one comparator. As mentioned in the Appendix the ACS logic computes and transmits two metrics using the largest metric of the two received metrics and one encoded bit of the input stream. A trellis diagram of a small viterbi decoder is shown in Fig. 4.

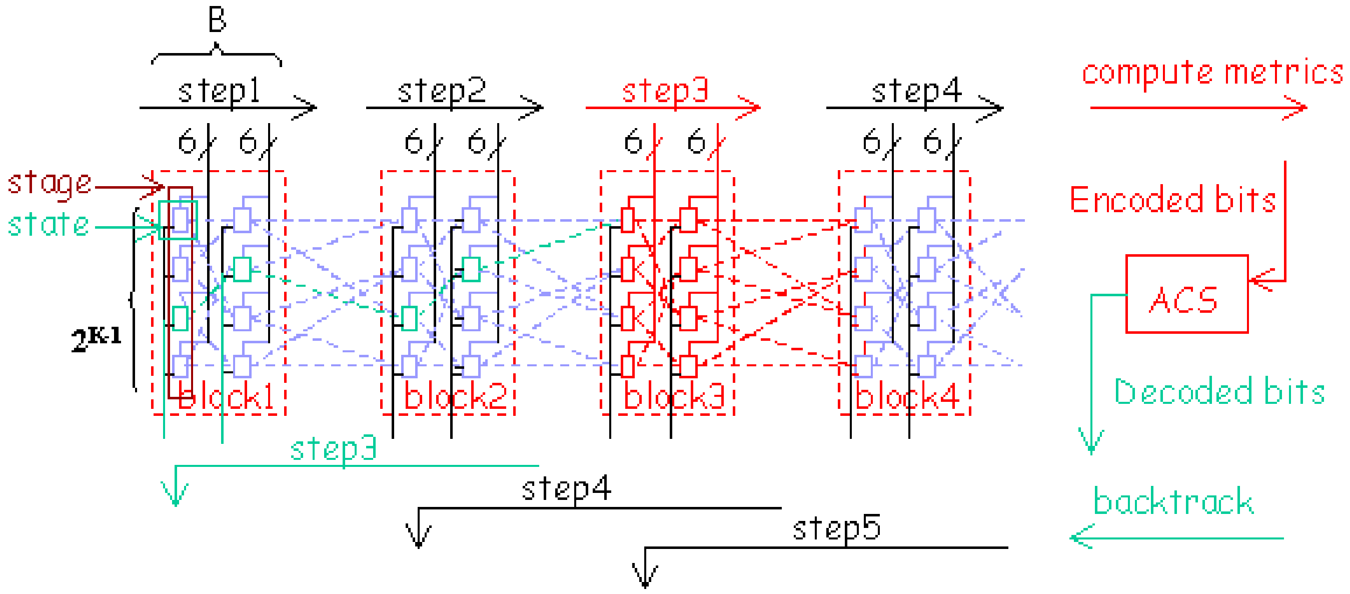

Figure 4: Operations performed by the Viterbi decoder

An ACS is called *"state"* in the diagram. A *"stage"* of an encoded bit consists of all the states that receive that encoded bit. Finally stages are grouped up into *"blocks"*. For example in the Figure there are 4 blocks with each block consisting of 2 stages and each stage consisting of 4 states. The input encoded bits arrive one after the other and the metrics of the stages are computed in the same order. In the Figure the input bits are received at the top, and the decoder gives the output decoded bits at the bottom. The dashed lines correspond to 2-way communications. A state sends the 8-bit metric that it computes to the states connected at its right during the *metric update* process, and can receive a 1-bit signal from one of its states connected at its right during the *backtracking* process.

The *metric update* process works as follows. A stage waits for the metrics from the stage at its left and for the encoded bit from the input stream to arrive. When both arrive all its states will compute their metrics and send them to the states they're connected to at their right. Thus, each state of the stage at the right will receive two metrics. In order to compute its own metric it will compare the two metrics that it received and keep the larger of the two.

The results from these comparisons, generated during the *metric update* process, are needed in order to generate the decoded bits. It turns out that in order to generate the decoded bits for Block N, we need to have the results from the metric comparisons of Blocks N and N+1. The process of generating these decoded bits from the comparison results is called *backtracking* since the states are visited in reverse order, i.e. starting from the rightmost stage of Block N+1 and finishing at the leftmost stage of Block N. In contrast to the *metric update* process where all states computed their metrics, in the *backtracking* process only one state of each stage is activated (in a way not described here). The decoded bits of Block N correspond to the comparison results of the states of this Block that were activated during the *backtracking* process. As we see whenever the *metric update* process computes the metrics of one Block (N+1 in this case), the *backtracking* process has to backtrack two Blocks (N+1 and N).

In Fig. 4 each step at the top represents the computation of the metrics of a block while each step at the bottom represents a backtracking step of two blocks. For example in the first step the decoder computes the metrics of the first block. In the second step the decoder computes the metrics of the second block. In the third step the decoder backtracks the first two blocks in order to find the decoded bits of the first block and at the same time computes the metrics of the third block. It continues in the same way in the next steps. The resulting operation of the Viterbi decoder is that at each step it computes the metrics of the next block and backtracks the last two blocks.

2.2 Parameters of the Viterbi decoder

The Viterbi algorithm consists of several parameters. The most significant are :

- The way the data stream is received. The encoded bits are transmitted as signed antipodal signals (i.e. 0 is transimitted with a positive voltage and 1 is transmitted with a negative voltage). They are received at the decoder and quantized with a n-bit quantizer. In the project the value of n is chosen to be 3. The quantized number is represented in 2's complement with a range of -4 to 3. The minimum value i.e. -4 represents bit value 0 while the maximum value i.e. 3 represents bit value 1. The decoder can get intermidiate values due to noise. This scheme is called 3-level *soft decision* and used by many Viterbi decoders.
- The *encoding rate* which is the ratio of the number of the decoded bits of the data stream over the number of the encoded bits. The encoding rate used is 1/2 which is used by the majority of the Viterbi decoders.
- The *constraint length* which is the number of bits that are used for encoding one bit. This is the number of the preceding bits in the input that are used for encoding the bit plus one (the bit being encoded). This parameter is called K in the report and the number of states in a stage must be 2^{K-1}.
- The *generator polynomials* which define how an input bit at the ViterbiA encoder is encoded. Of course we can't have the same polynomials for two Viterbi algorithms with different constraint length values.
- The number of *states per block*. This parameter is called B in this text.

In this report we study various Viterbi decoders by changing the constraint length (parameter K), the number of states per block (parameter B) and the states implemented in hardware. The last parameter is described in the next section.

It turns out that by increasing the constraint length or the number of states per block the Viterbi algorithm has better noise immunity.

3 The Architecture

We can imagine that the trellis diagram is implemented in hardware exactly as it is depicted in Fig. 4. Of course this would cost a lot. A more cost effective solution could be to implement in hardware a smaller number of stages (for example the stages included in one block) and reuse them in every clock cycle. This number is called H. The output metrics of the last stage can be stored in a register in order to be used by the first stage (of the next block) at the next cycle. The comparisons results of the states can also be held in registers that are used by the backtracking logic. This is illustrated in Fig. 5. Notice that B = H = 2.

The *forwarding* logic in the Figure includes the states of one block that compute the metrics of that block. In every cycle the metrics of one block are computed and the results from the metric comparisons are shifted in the register at the top. This shift register is called S1 in the rest of the report, while the one below S1 which is used by the backtracking logic is called S2. The shift register has to be large enough to hold the comparison results of the states of two blocks since the backtrack logic needs the results of two consequative blocks in order to find the decoded bits of the first one, as mentioned in the previous section. In every cycle the forwarding logic computes the metrics and comparison results of one block while the backtracking logic backtracks two blocks finding the decoded bits of the first one.

3.1 Scaling the number of stages per block (parameter B)

In this section we show the changes of the datapath when the number of stages per block of the Viterbi algorithm is modified. This is best described by an example. We will modify the Viterbi decoder of the previous section by changing the number of stages per block into 4 (B = 4, while keeping H = 2). The new datapath is depicted in Fig. 6.

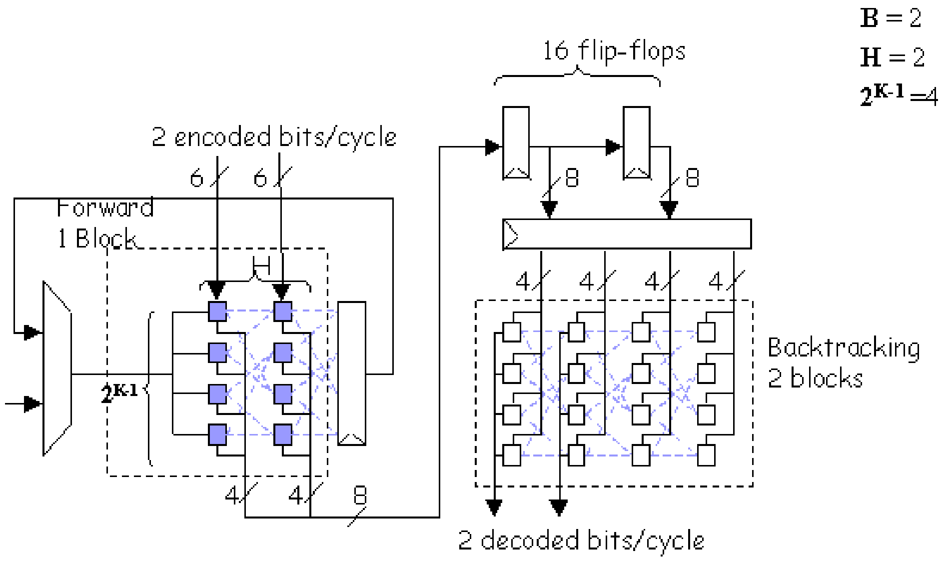

Figure 5: The Architecture of a small Viterbi Decoder

In the new datapath only a few registers have been added. Notice that now the forwarding logic needs two cycles to compute the comparisons results of one block, since B = 2 * H. Shift register S1 that holds the comparisons results must always be large enough to hold the comparisons results of exactly two blocks. Since the block has become two times larger the shift register S1 also became two times larger. The shift register S2 is used by the backtracking logic in order to find the decoded bits of one block. The register S1 copies its contents to register S2 every time that the comparisons results of one new block have been shifted in S1. While the forwarding logic shifts the results of one block in the shift register S1 the backtracking logic uses the shift register S2 which has the comparisons results of the last two blocks. During the cycles spent by the forwarding logic to shift the comparisons results of one block in the register S1, the backtracking logic backtracks two blocks using shift register S2. Intuitively the backtracking logic should be two times faster than the forwarding logic since the forwarding logic forwards one block the time the backtracking logic takes to backtrack two blocks. That means that the number of stages in the backtracking logic must be two times the number of stages in the forwarding logic.

3.1.1 Results of increasing the number of stages per block

Based on the previous observations we can find out how the number of stages per block influences the characteristics of the Viterbi decoder :

- It's obvious from the previous section that the hardware utilization increases slightly by increasing the size of the block (parameter B) since only a few registers are added.
- In order to find the speed that the Viterbi decoder operates we have to find the speed of the forwarding logic, since this determines the speed at which the decoder can process the input stream. The speed of the forwarding logic is determined by parameter H (stages implemented in harware) and the clock period since in every cycle the forwarding logic computes the metrics of H stages. Parameter H is equal to two in both two previous cases and is not influenced by parameter B. The clock period is not influenced by parameter B either since the clock period is determined by the critical path of the the forwarding and the backtracking logic which do not change. Hence the speed of the forwarding logic and therefore the speed of the Viterbi decoder stays approximately constant.
- Noise elimination gets better with increasing number of stages per block.

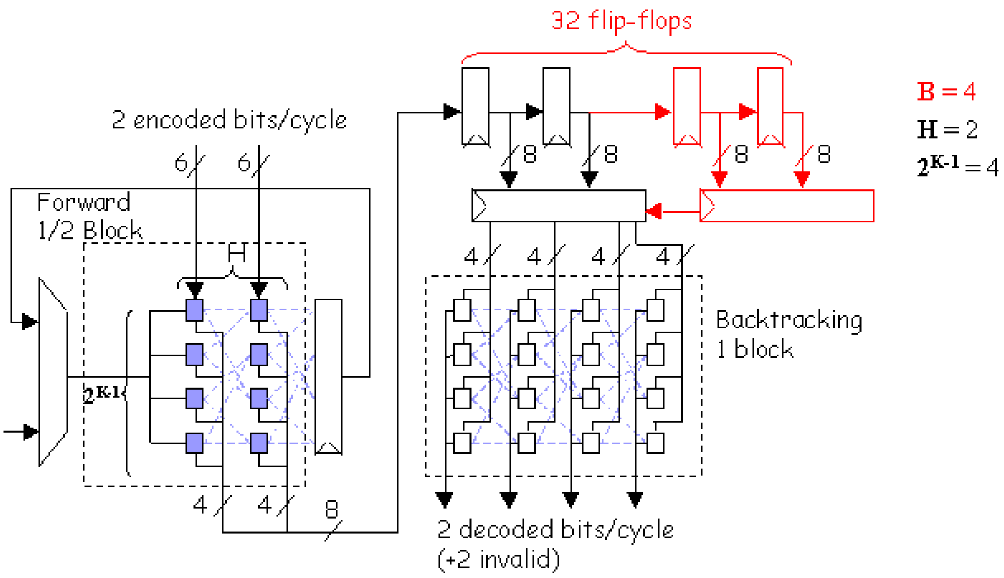

Figure 6: The new datapath with B = 4

So we can derive from the above observations that we can trade off hardware utilization for noise elimination. Actually, since the hardware is only slightly increased, we can afford big values of B.

3.2 Scaling the number of stages implemented in Hardware (parameter H)

In the previous section we showed a Viterbi decoder with $H < B$. Thus, in effect we have shown how the number of the stages implemented in hardware can be less than the number of the stages of one block. That means we have given a first description of how to use parameter H independently of parameter B (as long as $H \leq B$; we will see later in the simulation results section that we never need $H > B$ because cost increases a lot while effectiveness and speed remain almost the same). This section describes the architectural changes when scaling parameter H. As before this is best described by using an example. We change the parameter H, of the Viterbi decoder of the previous section, into 1. The new datapath is depicted in Fig. 7.

The number of stages of the forwarding logic is equal to 1, since H = 1. The number of stages of the backtracking logic must be two times the number of stages in the forwarding logic as we said in the previous section, thus there are 2 stages in the backtracking logic. The capacitance of the two shift registers has remain the same because each one must be large enough to hold the results of two blocks and the block size (parameter B) has not changed. Notice that the forwarding logic computes the metrics and comparisons results of one stage every cycle. Since a block consists of four stages (B = 4) the forwarding logic forwards one fourth of the block in every cycle and thus needs 4 cycles to compute the metrics of one block.

3.2.1 Results of decreasing the number of stages implemented in Hardware

From Fig. 7 we can derive how the number of the stages that are implemented in Hardware influence the characteristics of the Viterbi decoder :

- The hardware utilization is decreased significantly since both forwarding and backtracking logic halve while the number of flip-flops stays the same.
- The speed of the Viterbi Decoder is determined by parameter H and the clock period as shown in section 3.1.1. As we halved parameter H (from 2 to 1), we doubled the number of clock cycles needed to compute the metrics of one block. On the other hand the clock cycle almost halved since

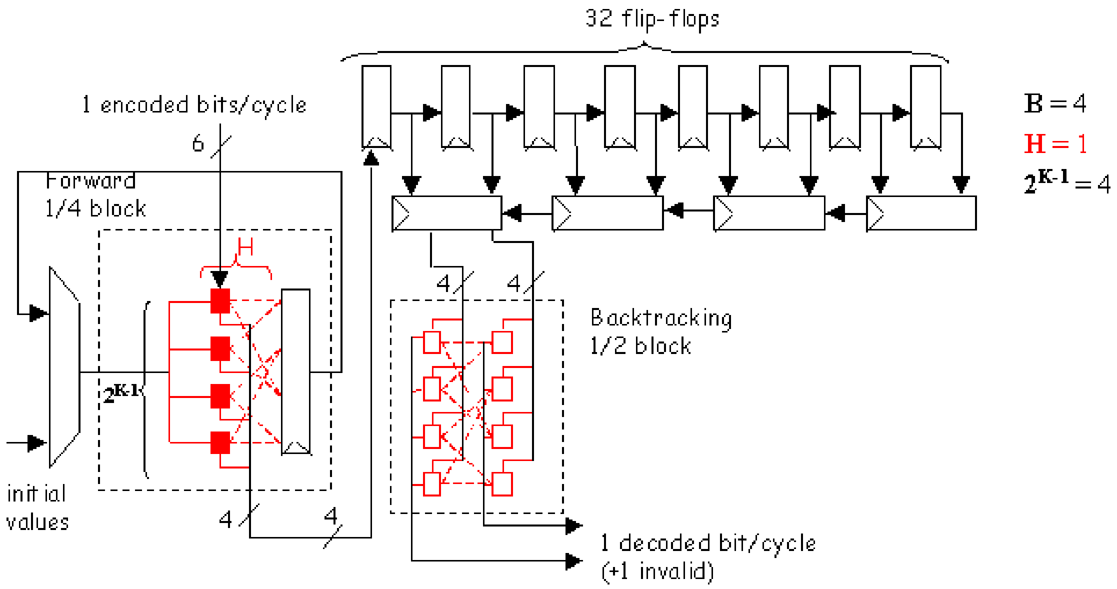

Figure 7: Changes in the datapath when H = 1

the forwarding logic became half than what it was before. For this reason speed slighlty decreases, but not much.

- Noise elimination remains the same since the Viterbi algorithm remains the same (B and K have not changed).

From the above observations we conclude that we can trade off speed for hardware utilization. Since speed is slightly decreased while hardware is significantly decreased it seems to be better to keep parameter H small.

4 Simulation in Verilog and Results

4.1 Simulation Process

In order to verify the observations of sections 3.1.1 and 3.2.1 and get accurate results several Viterbi decoders were implemented and studied. A Perl script was written in order to automatically generate the Verilog description of a Viterbi Decoder with certain parameters. This Perl script and some generated Viterbi decoders are available at http://www.cs.berkeley.edu/˜ iakovos/cs252/viterbi. The parameters of the Viterbi decoder that can be specified to the Perl script to generate a decoder are the ones mentioned in section 2.2. The resulting Viterbi decoders were compiled on the ALTERA's EPF10K100BQ208 FPGA using MaxPlus II. This FPGA has 4992 logic cells (it also has an internal memory consisting of 12 EABs but it was not used). The Verilog code generated by the Perl script is compatible to MaxPlus II. Hardware utilization and clock period (and hence speed of the Viterbi decoder) ware derived using MaxPlus II.

4.2 Results scaling the number of stages per block (parameter B)

To measure the influence of the number of stages per block to the FPGA utilization and speed of the Viterbi Decoder all the parameters of the decoder were kept fixed apart from parameter B. The Viterbi decoders that were simulated in MaxPlus II for these measurements have 8 states per stage (K = 4) and one stage implemented in hardware (H = 1).

4.2.1 FPGA Utilization

Figure 8 shows the FPGA utilization as the number of stages per block increases. We can see that the utilization increases as the number of stages increases. Increasing the number of stages by a factor of three results in increasing the FPGA utilization by a factor of two. In larger Viterbi decoders (with $K > 4$) we observe a larger difference in the increasing factors. The increase in utilization is due to the shift registers as described in section 3.1. So it is worth trading off this small FPGA utilization increase for more stages in the block (which provides better noise elimination).

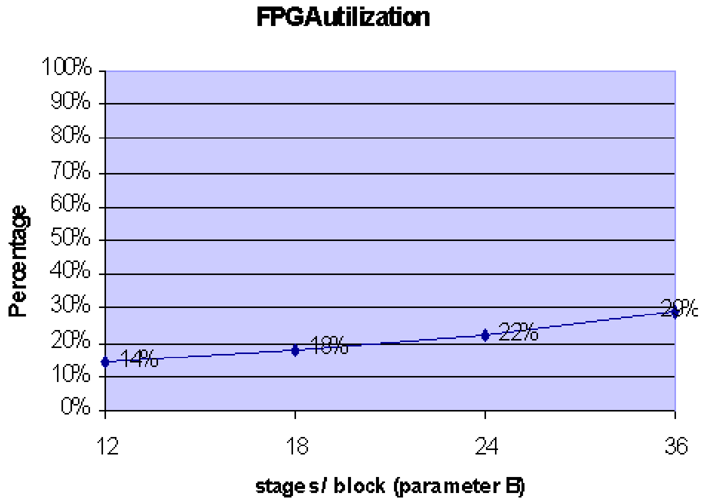

Figure 8: Number of stages per block versus FPGA utilization

4.2.2 Speed

Figure 9 shows the speed of the Viterbi Decoder as the number of stages per block increases. We can see that the speed is not influenced by parameter B, and remains almost constant even if the number of stages increases by a factor of 3. The differences between the 4 simulation results shown in the Figure are insignificant and are due to the placement and routing algorithms of MaxPlus II; it is reasonable not to get exactly the same results since the simulated circuits are not exactly the same.

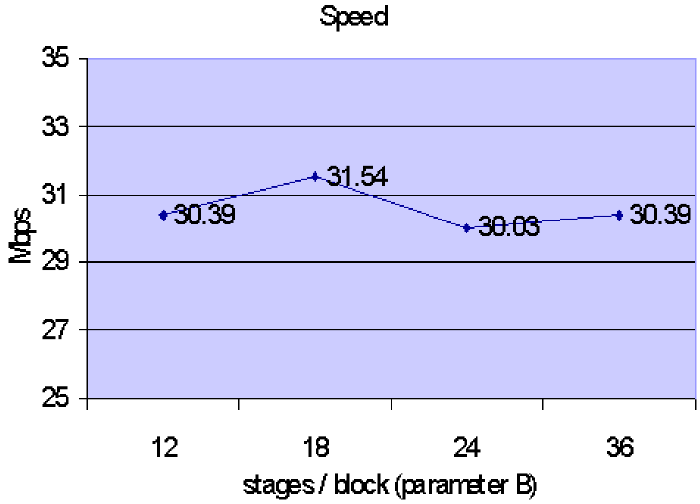

Figure 9: Number of stages per block versus speed

4.3 Results scaling the number of stages implemented in Hardware (parameter H)

In order to study the influence of the number of stages implemented in hardware to the FPGA utilization and speed of the Viterbi decoder we implemented several Viterbi decoders with constraint length 3 (K = 3), 8 stages per block (B = 8) and a varying number of stages implemented in hardware (parameter H).

4.3.1 FPGA Utilization

Similarly we can observe in Figure 10 that the FPGA utilization increases as the number of stages that are implemented in hardware increases. However now the FPGA utilization increases almost linearly. Inreasing H by a factor of four results in increasing the FPGA utilization by the same factor. If the decoder is large enough (for example if $K > 7$) this may be prohibitive.

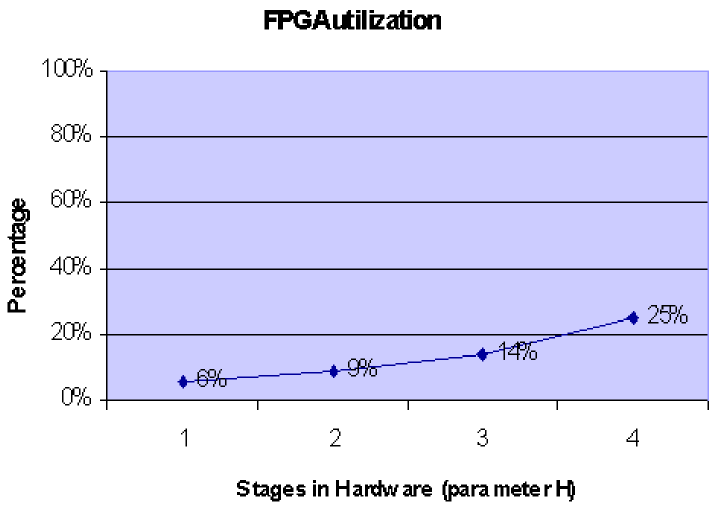

Figure 10: Number of stages in hardware versus FPGA utilization

4.3.2 Speed

The speed of the Viterbi decoder increases as the number of stages that are implemented in hardware increases as Figure 11 shows. This increase becomes smaller and almost insignificant as parameter H becomes larger. The maximum possible speed is obtained if we implement all the stages of the trellis diagram of Fig. 4 in hardware. It turns out that we can get the same speed for a smaller number of stages implemented in hardware. In the Figure the maximum potential speed is near 60 Mbps, and is almost reached for H = 3. Since the FPGA utilization increases linearly as we increase parameter H a value of $H < 4$ is preferable for this Viterbi decoder.

Hence by changing parameter H we can trade off hardware utilization for speed. The main conclusion of this section is that large values of H should be avoided since a small value can give a speed close to the maximum one while large values increase hardware utilization a lot .

4.4 Results scaling the number of states per stage (parameter K)

In order to operate well the number of stages of the block of the Viterbi decoder must be at least equal to four times paramater K minus 1. That means that $B \geq 4 * (K - 1)$ must hold. The Viterbi decoders studied to understand the influence of parameter K to speed and hardware utilization have one stage implemented in hardware (H = 1) and the number of stages per block equals four times parameter K minus 1 (B = 4 * (K - 1)).

Figure 11: Number of stages in hardware versus speed

4.4.1 FPGA Utilization

The FPGA utilization increases as the number of states per stage increase as Figure 12 shows. We can observe a steep increase as parameter K becomes larger. The output files of MaxPlus II show that a large portion of the hardware is dedicated to registers, and especially the register at the output of the forwarding logic which holds the metrics of the states of the last stage. Instead of using a large register to hold these metrics we could use the internal SRAM of the FPGA which would decrease the FPGA utilization a lot, but unfortunately we didn't have the time to do so. So we believe that the results of Figure 12 and especially the last one (i.e. 77% of the FPGA) are misleading and increase in parameter K does not necessarily lead to such a significant increase in hardware utilization.

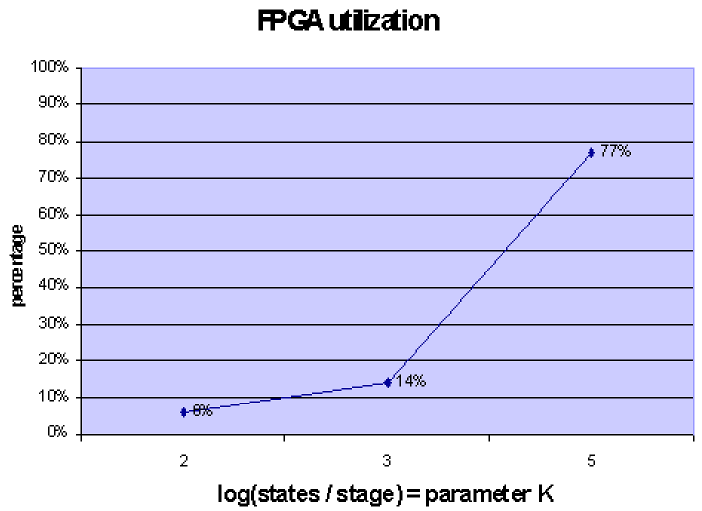

Figure 12: Number of states per stage versus FPGA utilization. The 3 simulated Viterbi Decoders have 4, 8 and 32 states per stage

4.4.2 Speed

In Figure 13 we can see that the speed of the decoder decreases as the number of states per stage increases. The speed decreases slightly and almost insignificantly as parameter K becomes larger. The

Viterbi decoders operate in a speed near 30 Mbps.

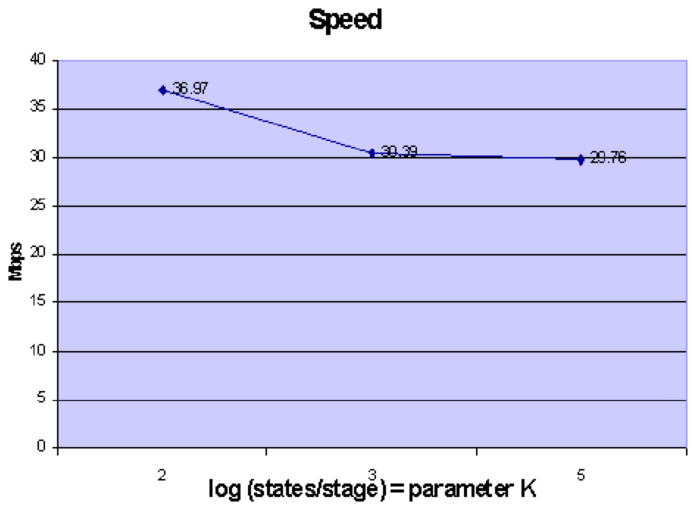

Figure 13: Number of states per stage versus speed. The 3 simulated Viterbi Decoders have 4, 8 and 32 states per stage.

The conclusion from this section is that the hardware increases significantly while the speed decreases slightly as parameter K increases. Since we have better noise elimination as parameter K increases we can trade off hardware utilization and speed for noise elimination. The hardware utilization would not increase as fast as it is depicted in Figure 12 if the internal SRAM of the FPGA had been used, and hence if the internal SRAM is used it is worth increasing parameter K.

5 Comparison of the implemented Viterbi Decoders with a Scalar core implementation

In this section the simulated results are evaluated comparing them with the results of a scalar core implementation of the Viterbi decoder. A scalar core implementation needs around 6 instructions per state for implementing the forwarding logic and another 3 instructions per stage for implementing the backtracking logic (that means it needs $6 * 2^{K-1} + 3$ instructions per stage). In the previous section a Viterbi Decoder with 32 states (K = 6) per stage operates at 29.76 Mbps. This is equivalent to a scalar core executing $6 * 2^5 + 3 = 195$ instructions per $1/29.76 = 0.0336us$, i.e. 5803 MIPS.

6 Related Work and Comparison with it

In the literature there has been much interest in implementing the Viterbi decoder using FPGAs the last 5 years. The recent related works reveal that the implementation of the Viterbi decoder on FPGA is certainly a contemporary issue.

Comparing the results of previous works with the ones presented in this report created some questions. In [2] the authors have implemented a Viterbi decoder with constraint length 7 using 4 XILINX 4028EX FPGAs. In [3] a Viterbi decoder with constraint length 9 and speed near 19.2 Kbps is presented. In [4] another Viterbi decoder with constraint length 9 and speed 32Kbps is implemented using 2 ALTERA Flex81500 FPGAs. Finally in [5] a Viterbi decoder with constraint length 14 and speed 41Kbps is implemented on 36 XILINX XC4010 FPGAs. The speeds of the above Viterbi decoders are in the Kbps range while the speeds presented in this report are near 30 Mbps. This large discrepancy is difficult to explain. Of course the Viterbi decoders implemented in this report are smaller (the maximum one has constraint length 6) and probably simpler. However that can't be a good explanation for the large difference in speeds. An in depth comparison between the work presented in this report and the previous ones may reveal the reasons.

7 Conclusion

In this work, we presented the implementation of the Viterbi Decoder on FPGA and studied area-speed-effectiveness trade-offs. Three parameters of the Viterbi Decoder permited us to trade off a characteristic of the decoder for another. In particular the following are some significant conclusions of this work: First the number of stages per block can be large enough since the area increases slightly, speed remains constant and effectiveness (noise elimination) increases. Second the number of stages implemented in hardware should be kept small since the maximum speed is reached with a small value while the area penalty of implementing more stages in hardware is big. Finally by increasing the number of states per stage the decoder becomes more effective but the hardware cost increases and speed slightly decreases.

8 Future Work

Some directions to continue this work are the following:

- The use of the internal SRAM of the FPGA in order to hold the metrics of the last stage of the forwarding logic is needed in the implementation of large Viterbi Decoders. When parameter K is larger than 6 the Viterbi Decoder does not fit in the FPGA unless the internal SRAM is used. Using the internal memory of the FPGA large Viterbi Decoders can fit in one FPGA.
- Another parameter that was not taken into account is the number of states per stage implemented in hardware. Probably this parameter has meaning only when one stage is implemented in hardware (H = 1) and we still want to decrease the hardware utilization. With the appropriate timing in hardware it is possible that the number of states in the forwarding logic is less than the states per stage of the trellis diagram (i.e. less than 2^{K-1}).
- Finally the implementation of more compex Viterbi Decoders using the proposed architecture can be studied.

9 Acknowledgements

First I would like to thank Professor Kubiatowitch for his overall guidance, remarks and encouragement. Also I would like to thank Ioannis Mavroidis for his significant help in this work and Thanasis Ikonomou for giving me access to MaxPlus II. Finally I would like to thank the authors of MaxPlus II.

References

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

[2] Kivioja, M.; Isoaho, J.; Vanska, L. *"Design and implementation of Viterbi decoder with FPGAs."* Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology, vol.21, (no.1), Kluwer Academic Publishers, May 1999. p.5-14.

[3] Pandita, B.; Roy, S.K. *"Design and implementation of a Viterbi decoder using FPGAs."* Proceedings Twelfth International Conference on VLSI Design, Goa, India, 7-10 Jan. 1999.) Los Alamitos, CA, USA: IEEE Comput. Soc, 1999. p.611-14.

[4] Jang-Hyun Park; Yea-Chul Rho *"Performance test of Viterbi decoder for wideband CDMA system."* Proceedings of the ASP-DAC '97. Asia and South Pacific Design Automation Conference 1997 (Cat. No.97TH8231), New York, NY, USA: IEEE, 1997. p.19-23.

[5] Yeh, D.; Feygin, G.; Chow, P. (Edited by: Pocek, L.; Arnold, J.) *"RACER: a reconfigurable constraint-length 14 Viterbi decoder."* Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines (Cat. No.96TB100063), Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1996. p.60-9.

APPENDIX: Operations of the Decoder

A short description of the decoder operations follows. The reader is also referred to [1] for a more thorough description. There are two kind of operations, metric update and bakctrack (or traceback).

- In metric update the following actions are performed: The node sends a metric to each node connected at its right. The closer the encoded bits are to the corresponding G0, G1 the larger the metric that is sent. Also, the larger the metric of a transition, the higher the likelihood that the resulting path will include this transition. So the metric that each node computes depends on how far away the received encoded bit pair is from the two possible transmitted pairs (the bits on the two transitions starting from this node). Each node receives two input metrics from the two nodes of the previous stage and selects the larger one to update with the metric it computes while the other one is discarded. The metric a node computes is added to the largest received metric and the result is sent to the next node. The computations of the metrics are done in order, i.e. all the states of a stage first receive the metrics from the previous stage and then send the new metrics to the next stage.

- In the backtracking the resulting path is determined. As mentioned in the metric update operation each node selects the larger input metric to update. The results of these comparisons of the ACS nodes are used by the backtracking process to determine the maximum likelihood path. Starting from a node at the last stage (the right most stage) the decoder backtracks the stages selecting the node at each stage that is included in the resulting path. This is done by backtracing the transitions whose metrics were selected in the metric update operation.

From the resulting path the decoded bits are derived by using the bits G0, G1 that correspond to the transitions of the path.