**Implementation of Area Efficient High Speed Viterbi Decoder**

**ADAM SHAFI SHAIR**, P. RAMESH, L. SRINIVASA RAO, K. SATISH BABU

1. Department of ECE, Shadan College of Engineering & Tech, Hyderabad, A.P., India.
2. GIT, GITAM University, Visakhapatnam, A.P., India.

Email: adamshafi1985@gmail.com, rameshpotlur@gmail.com, cnu.lam@gmail.com, id4babu@gmail.com

---

**Abstract:** The work done is based on a previous implementation that was analyzed in order to improve the already existing system performance (area, speed etc.). The Viterbi algorithm is a maximum-likelihood algorithm for decoding of convolution codes. The algorithm tries to find a path of the trellis diagram, where the sequence of output symbols approximately matches the received sequence. To accomplish this task, it calculates for each path, the path metric, which measures the distance to the received symbols sequence. The overall System will be designed using HDL language and simulation, synthesis and implementation (Translation, Mapping, Placing and Routing) will be done using various FPGA based EDA Tools.

**Keywords:** Convolutional Encoder, Trellis Diagram, Viterbi Algorithm, VLSI Design, FPGA.

---

**Introduction**

Viterbi decoder is one of the most widely used components in digital communications and storage devices. Although its software implementation is studied in depth over the last decades (matlab, embedded C) still every new design starts with design space exploration. This can be partially explained by the fact that the design space is huge. In addition, the optimization criteria and the design figures keep on changing with the advancement in semiconductor processing technology and design tools. Different design aspects of the Viterbi decoder have been studied in a number of research papers [1]-[7]. However, most researchers concentrate on one specific component of the design (e.g., path metrics unit or survival memory unit). Somewhat more general studies are presented in [1]-[9].

The proposed research work will carry out on reconfigurable FPGA technology, by adopting parallel/pipeline features of the hardware resources. The overall system performance could be improved. The exiting algorithm is redesigned using HDL language, simulation, synthesis and implementation (translation, mapping place & routing) done with FPGA based EDA tools.

By exploring the design at architecture, RTL level and selecting desired optimization constraints (area, speed, power) to Selected FPGA device synthesis technology, the system performance could be improved and comparison analysis could be carried out with old existing research works.

**Structure of Viterbi Decoder**

Viterbi decoding algorithm is the most popular method to decode convolutional error correcting codes. In a convolutional encoder, an input bitstream is passed through a shift register. Input bits are combined using the binary single bit addition (XOR) with several outputs of the shift register cells. Resulting output bitstreams represent the encoded input bitstream. The constraint length of the code is defined as the length of the shift register plus one. Finally, generator polynomials define, which bits in the input stream have to be added to form the output. An encoder is completely described by polynomials of degree or less. Fig. 1 shows an example of a simple convolutional encoder together with the corresponding parameters.

As can be easily recognized, a convolutional encoder forms a finite-state machine (FSM), whose state is described by the contents of the shift register. Every new input bit processed by the encoder leads to a state transition. One widely used presentation form of these state transitions is the so-called trellis diagram. An example of such a diagram is shown in Fig. 2. For a binary convolutional code, every state in the trellis has two incoming branches representing the transmission of one and zero, respectively.

Every state transition in the encoder results in one codeword being produced. After the transmission over a noisy channel, the code words may get corrupted. Viterbi decoder reconstructs the initial input sequence of the encoder by calculating the most probable sequence of the state transitions. This is done by tracing the trellis in a reverse manner while looking at the sequence of incoming codewords. Every codeword is produced as a result of a combination of a certain input bit with specific encoder state. This means, that the likelihood of the state transitions can be calculated even if received codewords are corrupted. By comparing the likelihood values, some transitions can be discarded immediately, thereby pruning the search space. Once the state transition sequence is determined, the reconstruction of the transmitted bit sequence is trivial.

At the top level, Viterbi Decoder consists of three units: the branch metric unit (BMU), the path metric unit (PMU), and the survivor memory unit (SMU).
The BMU calculates the distances from the received (noisy) symbols to all legal codewords. In case of the encoder represented in Fig. 1, the only legal code words are “00”, “01”, “10”, and “11”. The measure calculated by the BMU can be the Hamming distance in case of the hard input decoding or the Manhattan/Euclidean distance in case of the soft input decoding (e.g., every incoming symbol is represented using several bits).

The PMU accumulates the distances of the single codeword metrics produced by the BMU for every state. Under the assumption that zero or one was transmitted, corresponding branch metrics are added to the previously stored path metrics which are initialized with zero values. The resulting values are compared with each other and the smaller value is selected and stored as the new path metric for each state. In parallel, the corresponding bit decision (zero or one) is transferred to the SMU while the inverse decision is discarded. The structure of the add-compare-select circuit which performs the operations described above will be discussed in detail later.

Finally, the SMU stores the bit decisions produced by the PMU for a certain defined number of clock cycles (referred as traceback depth, TBD) and processes them in a reverse manner called backtracking. Starting from a random state, all state transitions in the trellis will merge to the same state after TBD clock cycles. From this point on, the decoded output sequence can be reconstructed.

Coding rate $1/n$, constraint length $K$, traceback depth $TBD$, coding rate and the number of bits representing each input value (referred as softbits or input bit width) are the key parameters influencing the performance and design of the Viterbi decoder. In the following sections, the affects of changing these parameters are discussed for every decoder unit separately.

**Branch Metric Unit**

BMU is typically the smallest unit of Viterbi decoder. Its complexity increases exponentially with (reciprocal of the coding rate) and also with the number of samples processed by decoder per clock cycle (radix factor, e.g., radix-2 corresponds to one sample per clock cycle). The complexity increases linearly with softbits. So, the area and throughput of the BMU can be completely described by these two factors.

As BMU is not the critical block in terms of area or throughput, its design looks quite straightforward. The version calculating the Hamming distance for coding rate $1/2$ presented in Fig. 1 performs perfectly in terms of both area and throughput. A BMU calculating squared Euclidean or Manhattan distance is slightly more complex but can be easily mapped to an array of adders and subtracters as well.

BM should be able to hold the maximum possible difference (distance) between ideal and received symbols. For hard input softbits, the Hamming distance between a coded word and its complement is the maximum possible BM (i.e., the distance between codewords consisting of all 0’s and all 1’s). BMs are generated assuming that every possible combination of bits is a valid encoder output. But as value beyond 4 is seldom used and radix is limited to factor 4, the area of the BMU is negligible compared to other units. For radix-8 designs, BMU will require considerably more area and also become slower.

**Branch Metric Unit**

In general, radix-8 Viterbi decoder designs are very uncommon due to their overall complexity. For radix factors beyond 2, combined with soft input decoding, BMU can also be implemented as a set of lookup tables or cascaded combination of adders and lookup tables. Depending on the cell library and the technology used, such implementations may occupy less silicon area than the straightforward approach. Still, taking into account the moderate...
area of the BMU, these area savings are almost invisible at the top level.

Path Metric Unit
PMU is a critical block both in terms of area and throughput. The key problem of the PMU design is the recursive nature of the add-compare-select (ACS) operation (path metrics calculated in the previous clock cycle are used in the current clock cycle).

In order to increase the throughput or to reduce the area, optimizations can be introduced at algorithmic, word or bit level. To obtain a very high throughput, parallelism at algorithmic level is exploited. By algorithmic transformations, the Viterbi decoding is converted to a purely feedforward computation. This allows independent processing of input blocks. The algorithmic parallel block processing methods intend to achieve unlimited concurrency by independent block decoding of input stream. These techniques result in quite high area figures. But as technological advancements are making the devices shrink, they are getting more attractive. Still, for a specific case, if required throughput can be achieved by utilizing word or bit level optimization techniques, there is no specific need to use algorithmic transformations.

Word level optimizations work on folding (serialization) or unfolding (parallelization) the ACS recursion loop. In the folding technique, the same ACS is shared among a certain set of states. This technique trades off throughput for area. This is an area efficient approach for low throughput decoders, though in case of folding, routing of the PMs becomes quite complex. With unfolding, two or more trellis stages are processed in asingle recursion (this is called look ahead). If look ahead is short then area penalty is not high. Radix-4 look ahead (i.e., processing two bits at a time) is a commonly used technique to increase decoder’s throughput.

A conventional fully parallel implementation of the Viterbi decoder implies using one radix-2 ACS per state of the trellis, processing one stage of trellis at a time. For decoders with very high throughput or very low area constraints, some optimization techniques at system level appear attractive. These techniques achieve area figures lower or throughput figures higher, than the conventional fully parallel implementation of the algorithm would provide.

Implementation
Trace forward (TF) technique [4] is used to eliminate the merge stage (i.e., the buffer referred as the “merge block”) which estimates the starting state in decode operation. This technique uses a set of registers that store the encoded state. In every clock cycle, the decision bits coming from the PMU are used to update these registers with new states corresponding to the previous trellis stage. After TBD clock cycles, all registers are expected to converge to the same state, which is at the same time the starting state for the decode operation.

Pre-traceback (PTB) [4] is a technique used to reduce memory access frequency. Instead of writing decisions at each stage processed by the PMU, stages are pre-traced and written as one composite decision. Thus, only one column of write operations is required for every trellis stages processed by the PMU. The pre-trace is implemented by a RE network of typically 3 or 4 stages.

Note that especially for small values of TBD, some trace-back techniques like trace-forward can show lower BER performance than the conventional trace-back.

All techniques mentioned above involve several accesses to the memory in the same clock cycle. This implies some design decisions to be made with respect to the memory architecture. Memory can be organized as multi-port or single port with increased access rate. In case the access rate is too high, several single port memories can be interleaved in a ping-pong manner. Particular memory architecture strongly depends on the actual design case (e.g., availability of the multi-port memory, maximum clock speed and width of the memory, etc.) All techniques mentioned above involve several accesses to the memory in the same clock cycle. This implies some design decisions to be made with respect to the memory architecture. Memory can be organized as multi-port or single port with increased access rate. In case the access rate is too high, several single port memories can be interleaved in a ping-pong manner. Particular memory architecture strongly depends on the actual design case (e.g., availability of the multi-port memory, maximum clock speed and width of the memory, etc.). Throughput and area figures for the SMU can be
calculated quite easily. For RE and TB, the critical path delay is determined by the flipflop toggle rate and SRAM access time respectively.

![Fig.4: simulation results](image)

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Flip Flops</td>
<td>290 out of 1,536, 18%</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>525 out of 1,536, 34%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logic Distribution:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of occupied Slices</td>
<td>350 out of 768, 45%</td>
</tr>
<tr>
<td>Number of Slices containing only related logic</td>
<td>350 out of 350, 100%</td>
</tr>
<tr>
<td>Number of Slices containing unrelated logic</td>
<td>0 out of 350, 0%</td>
</tr>
<tr>
<td>Total Number of 4 input LUTs</td>
<td>525 out of 1,536, 34%</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>17 out of 124, 13%</td>
</tr>
<tr>
<td>Number of GCLKs</td>
<td>1 out of 8, 12%</td>
</tr>
</tbody>
</table>

**Conclusion**

In this paper, a novel approach work will carry out on re-configurable FPGA technology, by adopting parallel/pipeline features of the hardware resources. The overall system performance could be improved. The exiting algorithm is redesigned using HDL language, simulation, synthesis and implementation (translation, mapping place & routing) done with FPGA based EDA tools and results are shown in the tabular forms. Although it’s VLSI implementation is studied in depth over the decades still every new design starts with design space exploration. This can be partially explained by the fact that the design space is huge. In addition the optimization criteria and the design figures keep on changing with advancement in CMOS technology and design tools.

**References**


