Power Optimized Viterbi Decoder Implementation Throught Architectural Transforms

João Portela
IST/INESC-ID
Rua Alves Redol N^o 9 Sala 134
1000-029 Lisboa, Portugal
jpop@algos.inesc.pt

José Monteiro
IST/INESC-ID
Rua Alves Redol N^o 9 Sala 136
1000-029 Lisboa, Portugal
jcm@inesc.pt

Abstract

Viterbi is an algorithm for error correction in the transmission of messages. It requires coding and decoding stages in the sender and receiver, respectively. These type of algorithms are very useful for the transmission of a type of messages where some degree of error in the received message is acceptable, such as, voice and video. The coding allows some error detection and correction.

In this paper we present an architecture for the Viterbi Decoder. Using this initial structure, we have applied a set of transformation techniques aiming for a power optimized implementation. These techniques include pipelining, operation reduction/substitution and the reduction of transition activity. We show that it is possible to reduce the circuit's power consumption by more than half without impacting excessively on the area.

1 Introduction

The Viterbi algorithm, proposed by Forney [5], brought many benefits in a large number of problems, such as filters, transmissions and error detection, etc. There has been some work in the implementation of Viterbi algorithms, for instance [6, 8, 2, 7]. For the majority of them, the architecture optimization targeted the detection and correction of a larger number of errors. In this work we propose the use of transformation techniques to reduce the power consumption in a simple viterbi decoder architecture. These techniques were applied on a Viterbi decoder for the rate 1/2 convolutional code.

The developed architecture has the capacity to decode messages 8-bit long. This constraint is due to the faster/easier implementation, and it does not change any of the optimizations introduced.

The implementation was made in the BLIF format. This format was used because it is supported by SIS [9] whose power estimation tool we used in this work. Initially we implemented a Viterbi Decoder circuit without any low-power concerns. On this initial structure we tested some methods in order to reduce the power consumption. The methods used will be described together with their results. In addiction to our own ideas, we experimented with methods described in [4], [1] and [3].

This paper is organized as follows. In Section 2 we describe both the coder and decoder algorithm. The implementation is described in Section 3. In Section 4 we describe all the optimizations tested and their results. The conclusions of this paper are in Section 5.

2 Viterbi Algorithm

The Viterbi Decoder is similar to the coder method but works backwards. The coder algorithm is explained first.

2.1 Viterbi Coder

The Viterbi algorithm is based on a 2^n state algorithm, where n is the number of bits that is transmitted for each bit in the original message. Hence, if the message has m bits the Viterbi Coder sends $m \times n$ bits. In each step of the algorithm there is a single state, between 0 and $2^n - 1$. Initially the state is 0.

The bits sent are generated by a state machine that functions as follows. For the present state and bit to transmit we have to generate the n bits to send and the next state. The values used can be displayed in a table or a diagram. An example of a diagram for $n = 2$ is shown in Figure 1. We follow this state machine until all bits have been sent.

2.2 Viterbi Decoder

Viterbi Decoder is similar to the Viterbi Coder, but it presents some differences. One is that all the 2^n sequences

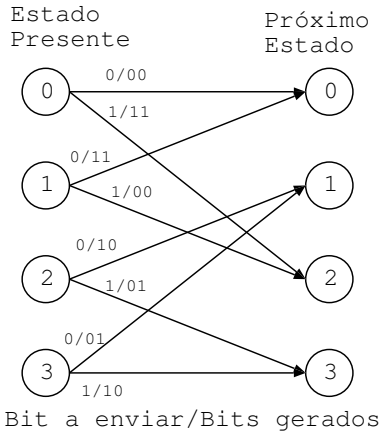

Figure 1. State diagram showing for each bit to send how the next state and bits actually sent are generated in the Viterbi coder.

with possible messages are kept during decoding and a function is used to compare those different sequences. The 2^n sequences are needed because the algorithm tries to minimize the error between the sequence received and the one generated.

At each step of the algorithm are created 2×2^n new sequences based on the last 2^n messages (corresponding to the 2^n possible states). It is created one message for each possible received bit, *i.e.*, 2 sequences are generated for each state $2 \times (2^n)$, see Figure 1. Now we have 2 sequences for each possible next state, and the decoder must select one of them. These 2 sequences are compared with a function, which is called "metric". This metric function returns a distance between the messages. The one that is selected is the one that presents less error, *i.e.*, with lower metric. At each stage the value of the metric can be saved in order to help future calculations. At the end is chosen the message with lower metric from all the possible 2^n sequences.

In this work we used the Hamming distance (sum of the number of bits that are different) as our metric function.

As has already been said we need to keep the 2^n entire sequences so we can compare them and make the most correct choice. Because of memory limitations in our hardware implementation we use slices of 8 bits and treated them separately.

3 Hardware Implementation

The hardware implementation has three distinct blocks: one to control the operations, one to execute the decode and another to choose the final result. Figure 2 shows the diagram for this architecture.

Next we explain with some detail all the three parts of the implementation.

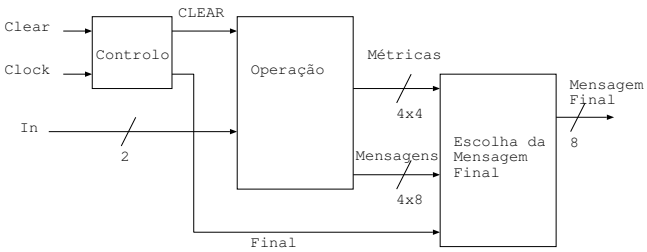

Figure 2. Hardware Diagram for the Viterbi Decoder.

3.1 Control Block

The Control block generates control signals to the Operation and Final Message Chooser blocks. It enables two signals called "CLEAR" and "Final" signals. The first clears all memories in the Operation block. The second one informs the last block that it is possible to decide on the decoded message.

Due to memory limitations, we have only kept 8-bit slices of the messages at a time. Thus, we have a 3-bit counter in order to indicate when the 8-bit message has already been decoded and a new 8-bit slice can start.

The "Final" signal is generated each eight clock cycles, with the help of the 3-bit counter. The "CLEAR" signal is generated by the existence of one of two signals, where the first is an external "Clear" signal and the other is the "Final" signal. The "Clear" signal is used to begin the decoding process. It should be observed that this implementation could be easily extended to higher order circuits by using a higher counter circuit.

3.2 Operation Block

The Operation block executes the decoding algorithm and it is composed of two main parts. The first part is the Memory block that keeps the sequences and their metrics. The second one creates the new sequences and the respective new metrics.

3.2.1 Memory Block

The number of latches used in this block depends on the number of bits used in the state algorithm. In our algorithm, we are using $n = 2$, and therefore we have 4 states and consequently 4 messages kept in the decoding algorithm. We have 8 latches for each sequence and 4 latches for each metric, thus we have a total of $48 = (4 \times (4 + 8))$ latches in the Memory block. In should be observed that this implementation can also be extended to higher sequences, using the correct memory extension.

3.2.2 New Sequence Generator Block

This block creates the 3 new sequences and their metrics. As mentioned in Section 2.2, for each old sequence 2 new sequences are generated. As there are 4 old sequences, 8 new sequences will be created but we have to choose only 4 of them. Figure 3 shows how the message for state 0 is created. The new sequence is created by the best choice of two sequences that are calculated based on the previous ones. This choice is based on the metric for the new sequences, which are also calculated in this block. As shown in Figure 3, the new metric and new sequence are available in the outputs block after some operations. One of these operations includes the Hamming distance calculation between fixed values (00 and 11), and the input subsequence. The values 00 and 11 correspond to the two alternative ways to arrive at next state 0 form the present states 0 and 1, respectively as show in the diagram of Figure 1. In this case, both of these states present the value 0 for the input bit. The results from the Hamming distance calculation are added to the previous metric values of states 0 and 1, in order to generate the metric of the new sequences. The lower of these values is the metric selected by the comparator circuit. As can be observed in Figure 3, this comparator is also used to choose the new sequence.

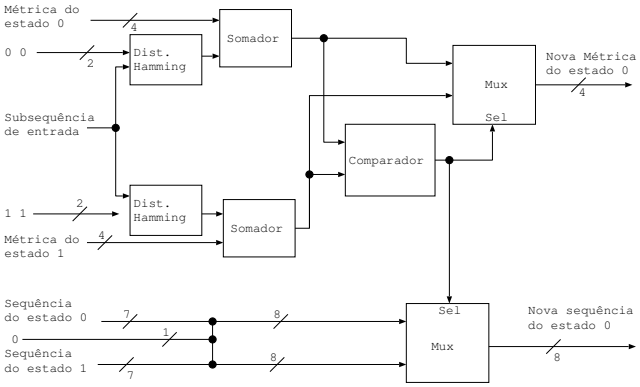

Figure 3. New Sequence Generator for message of state 0.

For the generation of the messages for the others states the same procedure is used. In this case it is only necessary to perform an adjustment in the input values of the current state. All of this information is obtained from the State/Bits diagram shown in Figure 1.

3.3 Final Message Chooser Block

This last block has only one objective, to choose from four possible sequences the one with the lowest metric value. All we need is a comparator to compare all four metrics and a multiplexer to pick the chosen sequence.

4 Optimizations and Results

In this section we explain all the techniques used and show the results obtained.

4.1 Optimizing the Final Message Chooser block

The block under optimization chooses the message decoded, so it is only used once in each eight clock cycles, because it is the time needed to decode a sequence of eight bits.

If this block was used as described in Section 3.3, it would choose a message in all clock cycles. It is obvious that this block spends more energy than necessary.

To avoid unnecessary power consumption we need to stop the input signals from propagating into the block. There are two choices, to use simple gates or use latches in the block's inputs.

Here we apply the first technique because the second will spend more since the latches spend more than a simple gate, such as an AND gate.

So we used for each input signal a gate AND with the "Final" signal. To study the impact of this technique we studied three places where to introduce the AND gates:

A - At the inputs of metric values

B - At the inputs of metric values and inputs of sequence 0

C - At all inputs

When AND gates are added in input metric values, only the multiplexer that choose the sequence will be active, choosing sequence 0. So the second experience was to stop the inputs of sequence 0 too, in order to avoid possible transitions in these signals. The last experience stopped all signals.

In the next sections we will use the terminology original, A, B and C circuit to refer to the type of Final Message Chooser block used in the experience.

Table 1 shows the results for these experiences. Till the B circuit there is a good improvement in power, though for the D circuit there is no real gain in comparison to the B circuit. The area increases in each new circuit, but only by 5.4%, in the worst case.

Note that for the original circuit the Final Message Chooser block spends 582.25μW and that in the B circuit it spends 120.45μW, *i.e.*, about 20% of the initial value! In the simulation with real delay the percentage of initial value is 12.3%.

With this simple modification we achieved power reduction of 12% with zero delay and 57% with general delay.

Table 1. Final Message Chooser Optimization.

Type of Simulation	Power (in μW)						
	Original circuit	A circuit		B circuit		C circuit	
Zero delay	4153.3	3691.5	88.9%	3649.2	87.8%	3678.3	88.5%
Real delay	25002.9	11072.6	44.3%	10804.2	43.2%	10705.3	42.8%
Area	2634	2682	101.8%	2706	102.7%	2778	105.4%

4.2 Optimizing Internal Sub-Blocks and Removing Constants

In this section we tried to optimize all the little details in all blocks and sub blocks. By little details, we mean signals that are not used, constant signals that can be removed, duplicated blocks, etc.

4.2.1 Removing Unused Signals and Constant Signals

When we are building the circuit we generally use pre-constructed modules, like adders, multipliers, etc. These are prepared for all possible cases, but in our application some of those cases are not used, like the C_{in} and C_{out} signals in adders. So all unnecessary output signals were removed and constant input logical signals (like a C_{in} not used) were simplified.

This technique was specially used in the adders, *i.e.*, we used a full 4-bit adder to add a 4-bit number with a 2-bit number, knowing that the output value is lower than 16. So we remove three inputs and one output in each adder.

Another application of this technique was in the memory block. Initially all 8 bits of the four sequences were stored, but for each state we know which is the last bit inserted and that the first one is not used, so these were also removed.

4.2.2 Removing Duplicated Blocks

Here we reduce some subcircuits that were performing the same job.

In our case, there were eight blocks to calculate the Hamming Distance between the input subsequence and the possible ones. But there are only four possible output sequences, because they have only 2 bits. This optimization used also the technique explained in the previous subsection, because the signal for all four possible subsequences (see Figure 1) are constants. So it was created a single subcircuit that calculates all four Hamming distances.

The results of these optimizations are in Table 2. The power reduces about 12% when comparing with the B circuit (with real delay),

which is a good result. The total area of the circuit is 30% lower, due to the reduction of many unnecessary gates.

Table 2. Changing internal sub-blocks and removing constants.

Type of Simulation	Power (in μW)		
	B circuit	B circuit + Sec.4.2 optimizations	
Zero delay	3649.2	2978.3	81,6%
Real delay	10804.2	9482.7	87.8%
Area	2706	1899	70.2%

4.3 Comparators Optimization

One of the techniques tried, was to reduce the comparator's power [1]. The objective was to avoid signal propagation of the lower bits of the comparator when the higher bits of the two numbers are different and there is no need to compare them.

In the article [1] latches are used to stop propagation, but in this case if we used latches we had to retime all circuit. So we tried to use AND gates, like in Section 4.1, to see if we could get some gain.

This technique did not result with the AND gates because the power reduction did not compensate for the power of the new gates. Table 3 shows the result of inserting the gates in the B circuit.

Table 3. Comparators Optimization.

Type of Simulation	Power (in μW)		
	B circuit	B circuit + Comparators Opt.	
Zero delay	2978.3	3067.9	103.0%
Real delay	9482.7	11512.1	121.4%
Area	1899	2025	106.6%

The power goes to 103% with zero delay and 121% with delay. This performance can be explained due the comparators only having 4 bits. If this number was greater it would have compensated.

With the delay simulation we have a worse result due to the fact that when the signal of the comparison between the first bits arrives at the gate, the other signal has already passed and will not be removed. So the number of transitions will be higher, and so will the power consumption.

Table 4. Pipelining.

Type of Simulation	Power (in μW)							
	Unpipelined circuit	Pipelined circuits						
	B circuit	B circuit	B circuit + Sec.4.2 optimizations		B circuit + latches		B circuit + latches + Sec.4.2 optimizations	
Zero delay	3694.2	6111.6	4756.1	77.8%	6406.1	104.8%	4980.6	81.5%
Real delay	10804.2	52730.8	45473.0	86.2%	20439.1	38.8%	17995.1	34.1%
Area	2706	4225	2739	64.8%	4305	101.9%	2819	66.7%

4.4 Pipelining

Pipelining is a powerful technique. I can be used to serialize procedures, raising its throughput. Here we tried to use pipelining also with the goal to reduce power consumption.

The first try was to place two New Sequence Generator block instead of one in the Operation block in order to perform 2 steps at a time. To do this we need to change the Control block because at each clock cycle we generate the result corresponding to two old clock cycles without pipeline, *i.e.*, we will have results in four clock cycles instead of 8.

These new circuits were tested using the optimizations of unpipelined circuits of Sections 4.1 and 4.2 and the use of latches between the two New Sequence Generator blocks (to reduce gate switching).

The results of these experiences are presented in Table 4. By comparing the B circuit without pipeline and the one with, (without 3649.2μW, with 6111.6μW) we can gain some power due the clock frequency reducing to half and the correct value of 6111.6μW is 3055.8μW. We can see that the area of the new circuit is not the double (we duplicate only the operation block). If we compare the simulation with delay, the results are not so good, this circuit dissipates 52730.8μW (half=26365.4μW) and the old 10804.2μW, *i.e.*, we do not have an optimization.

When we use the optimization of the unpipelined circuits ("Sec.4.2 optimizations"), the value drops to 4756.1μW (half= 2378.05μW). But with delay simulation we still have worse results.

The other type of optimization, the use of latches, tries to reduce glitching. Table 4 also shows these optimizations. Although the simulation with zero delay raises (comparing with the non-latches circuits), the simulation with the delay model gets better results, the power drops around 61%. The latches were placed after the first metrics calculation. Comparing the consumption of the "best" circuit (with delay simulation), 17995.1μW (half=8997.55μW), with the unpipelined 10804.2μW, we achieve better results.

The area of the final circuit (with pipeline) is similar to the first circuit made.

This simple example shows that the decoding can be pipelined at more levels with a good power reduction. It depends on the area that the designer is willing to use.

4.5 Algorithmic Exploration

Some architectural exploration has been made, but with "few" results because in this circuit (Viterbi Decoder) the next states are very dependent on the last ones. Some tries were to minimize metric calculation, when using a simple pipeline, as follows.

To calculate the metric for sequence of state 0 at time $t+2$, it depends on the metrics of the sequences of states 0,1,2 and 3 at time t, as can be seen in Equation 4.5.

$$Metric'(0,t+2) = \begin{cases} Metric(0,t) + Ham1(00) + Ham2(00) \\ Metric(1,t) + Ham1(11) + Ham2(00) \\ Metric(2,t) + Ham1(10) + Ham2(11) \\ Metric(3,t) + Ham1(01) + Ham2(11) \end{cases}$$

Where $Ham1()$ and $Ham2()$ represents the Hamming distance between the received bits and the input bits, for time t and $t+1$, respectively, *i.e.*, calculate the metric without the intermediate metric. However this alternative requires 3 comparators, 8 adders (4 plus 2 bits) or 4 adders (4 plus 2 plus 2 bits) and 3 multiplexers per state. Summing up: 12 comparators, 24 or 12 adders and 12 multiplexers, but in the original we have 8 comparators, 8 adders (4 plus 2 bits) and 8 multiplexers, for metric calculation. And we also need bigger multiplexers to calculate the new sequences. Therefore no power improvement were possible.

4.6 Operand Coding

Operand codification was another method we tried in order to optimize power. Gray and Hybrid [4] coding were tried in the codification of the metric values.

4.6.1 Gray Coding

One of the most promising encodings that is used to reduce switching activity is the Gray code since only one bit changes between consecutive values. Gray encoding is an excellent technique used in sequential numbers. For example sequential accesses at memory.

Table 7. Conclusions Table.

Type of Simulation	Power (in μW)								
	Unpipelined circuit			Pipelined circuit			Unpipelined vs Pipelined circuit		
	Original circuit	B circuit + Sec.4.2 optimizations		B circuit	B circuit + latches + Sec.4.2 optimizations		UnP. circuit B + Sec.4.2 opt.	Pip. circuit B + latches + Sec.4.2 opt (divided by 2)	
Zero delay	4153.3	2978.2	71.7%	6111.6	4980.6	81.5%	2978.2	2490.3	83.6%
Real delay	25002.9	9482.7	38.0%	52730.8	17995.1	34.1%	9482.7	8997.6	95.9%
Area	2634	1899	76.1%	5225	2819	66.7%	1899	1410	74.2%

Table 5. Number of Gates for Different Codings.

Operator	Binary	Gray	Hybrid
Adder 4+2	10	36	15
Comparator	13	23	15
Hamming Dist.	8	9	9
Total adders(8)	80	288	120
Total comp.(7)	91	161	105
TOTAL	179	458	234
circuit Total	519	789	574

Table 6. Power for different Coding (with B circuit).

Type of Simulation	Power (in μW)				
	Binary	Gray		Hybrid	
Zero delay	2978.3	5159.3	173.2%	3426.7	115.1%
Real delay	9482.7	13055.5	137.7%	13104.8	138.2%
Area	1899	2324	122.3%	1901	104.3%

The worst about Gray is its hard implementation. This fact is due to dependences of the output bits, *i.e.*, all output bits depend on all input bits.This can be seen in Table 5.

In the total Gray coding has almost 2.5 times the number of gates of binary coding, so the power dissipation, which is displayed in Table 6, was already expected to be higher. Also, in this application the numbers are not very sequential.

4.6.2 Hybrid Coding

The Hybrid coding is a mix between binary and Gray. It has proprieties from both encoding. Its implementation is easier than Gray's, but a little harder than binary. Although power dissipation raised with this implementation, Table 6, it can be useful when using large multipliers, see [4].

5 Conclusions

The architecture proposed is very simple and can be easily modified to support larger messages. The optimizations introduced in the circuit reveals a reduction of 62% without the use of pipelining technique, 66% when we use a 2-level pipeline, and 4% comparing the version with pipeline and the one without. These results are presented in Table 7. Note that all the optimizations realized were very simple. They tried to minimize unused signals and the logical gates associated.

It was also tried techniques to reduce the consumption, such as different types of number coding, comparator's optimization, with no success.

Area values have been reduced because the optimizations used in Section 4.2 were more efficient than the ones used in Section 4.1.

Acknowledgments

This research was supported in part by the portuguese FCT under program POCTI.

References

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-Based Sequential Logic Optimization for Low Power. In *Proceedings of the International Conference on Computer-Aided Design*, pages 74–81, Nov. 1994.

[2] S. Augsburger and C. Savarese. 3g turbo decoder, ee225c. Technical report, UC Berkeley, 2000.

[3] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodensen. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):12–31, Jan. 1995.

[4] J. M. E. Costa and S. Bampi. Power optimization using coding methods on arithmetic operators. *IEEE International Symposium on Signals Circuits and Systmes (accept for publication)*, July 2001.

[5] G. Forney. The viterbi algorithm. *Proc. of The IEEE*, 61(3):268–278, Mar. 1973.

[6] D. Garrett and M. Stan. Low power architecture of the soft-output viterbi algorithm. *ISLPED98, International Symposium on Low Power Eletronics and Design*, 1998.

[7] R. Pacheco, A. Susin, and L. Carro. Implementação de macrocélula do algoritmo viterbi. Article from UFRGS, Brazil (in Portuguese).

[8] B. Pandita and S. Roy. Design and implementation of a viterbi decoder using fpgas. *International Conference on VLSI Design*, 1999.

[9] E. Stentovich et al. *SIS - A System for Sequential Circuit Synthesis*, May 1992.