Chapter 6.

Optimized Viterbi Decoder Architectures

Contents

With the ever increasing demand for low-power battery-powered devices, power reduction techniques become more and more necessary throughout the design flow. One of the solutions to the problem of power reduction is to create new algorithms or to modify existing ones to take advantage of the run-time changes in the operating conditions or performance requirement, and adjust their behavior accordingly in order to lower their power consumption. Some applications in this direction can be found in [48] and [26].

In this chapter we address the Viterbi algorithm, proposing an architecture for the trace-back unit, whose size can be adjusted dynamically by adapting itself to the run-time variations in channel quality and performance requirements, together with an algorithm for finding the optimum trace-back length. Another interesting solution for dynamic power reduction in a Viterbi decoder can be found in [32].

As part of our research efforts regarding the realizability of low-cost digital radio receivers on a single FPGA, we also present a very low-area solution, in light of the new features offered by modern devices. For such low-rate applications, speed is not usually the main concern, the main goal being rather to minimize the necessary FPGA resources (or silicon area for ASIC's). Together with the FFT, the Viterbi decoder is the most computationally intensive

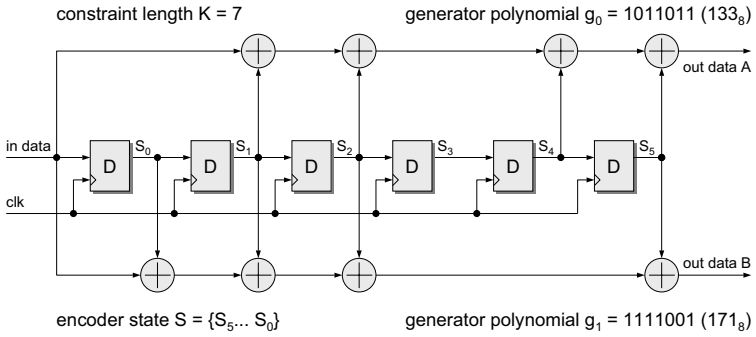


Figure 6.1.: Convolutional encoder for the DAB & DRM standards

block in an OFDM receiver and requires the most resources, being therefore a natural candidate for optimizations. In this thesis, we propose a low-complexity Viterbi decoder architecture optimized for FPGA that takes advantage of the embedded RAM blocks. The design has been implemented in VHDL as a completely generic RTL description. FPGA implementation results show that the proposed architecture requires extremely few hardware resources, while meeting the throughput requirements of the DAB and DRM standards.

6.1. Viterbi Decoding Background

Proposed in 1967 [93], the Viterbi algorithm rapidly became the solution of choice for decoding convolutional codes. It consists in finding the most likely state sequence through the trellis obtained by unrolling the state transition diagram of the encoder. The convolutional encoder is a state machine that generates a multi-bit output symbol based on the current input and the current state. **Figure 6.1** shows the schematic of the encoder for the DAB and DRM standards. Both standards use the same convolutional code. The octal forms of the four generator polynomials are: 133, 171, 145 and 133.

The two main parameters of such a convolutional encoder are the constraint length K and the number of branches or generator polynomials N. The number of states of the encoder is thus $S = 2^{K-1}$, increasing exponentially with the constraint length. For example, the convolutional code for DAB and DRM has a constraint length of 7, which results in 64 possible states. Every input bit determines the transition to another state, while generating an N-bit symbol. A state transition diagram (trellis) for $K = 3$ is shown in **Figure 6.2** for six successive symbols.

From any given state, two other states that differ only in their LSB can be reached (an odd/even pair). Each state pair can be reached from another pair, whose states differ only in MSB. Such a state transition butterfly is shown in **Figure 6.3** for $K = 7$. One important remark is that although there are four different transitions, only two different output combinations are produced, out0 and out1, which allows for simplifications in the decoder.

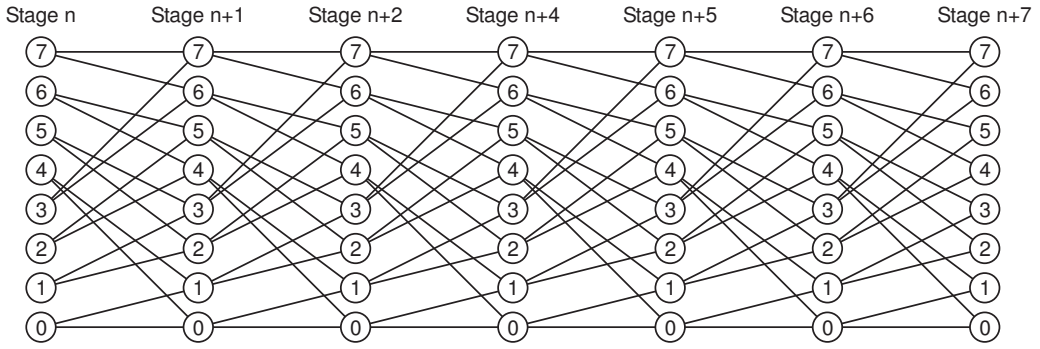


Figure 6.2.: State transition graph

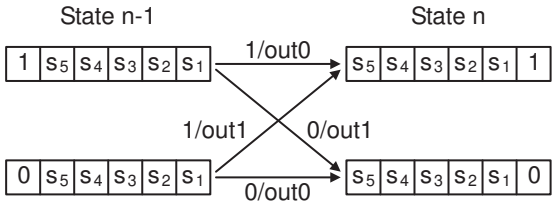


Figure 6.3.: Possible state transitions

On the receiver side, the decoder's task is to reconstruct the state sequence of the encoder, thus obtaining the original bit sequence as decoded output. The decoder only finds the maximum likelihood sequence of states that could have produced the received data stream. Each state transition is associated a branch metric, which is an indicator of the likelihood of the transition for the current received symbol. Since the current state is not known, all possible branch metrics have to be computed, based on the received probabilities (soft bits). In our implementation, the soft bits are positive integers, encoding probabilities between 0 and 1. Complementing a probability is obtained by negating all bits (one's complement). The decoder receives N soft bits per symbol and computes 2^N branch metrics. The parallel computation of all possible branch metrics is shown in **Figure 6.4** for $N = 4$ (DAB and DRM).

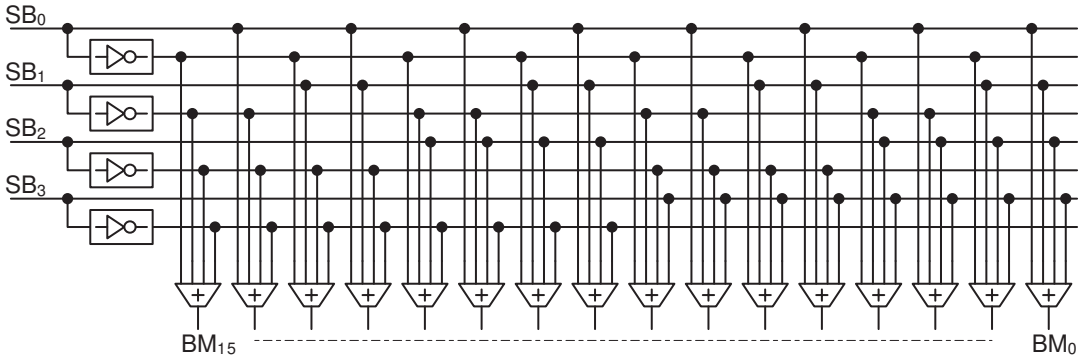


Figure 6.4.: Parallel branch metric computation

The decoder maintains a path metric for each state, which is a measure of the likelihood of that state to be the actual encoder state that produced the received symbol. Whenever a new symbol is received, new path metrics are computed for each state, based on the previous path metrics and the current branch metrics. Since two possible paths converge to each state, a decision is made by retaining only the more likely path, i.e. the path with a better metric, while the other is discarded. In our design, the convention is that lower metrics are better. This binary decision is all that is needed for determining the previous state from the current state (shift right and append decision bit as MSB). For each state, the path metrics of the two candidate states are calculated by adding the appropriate branch metrics to the previous path metrics. Only the best metric is selected and stored in the path metric memory, while the decision bit is stored in the trace-back unit. A number of 2^{K-1} such add-compare-select (ACS) operations have to be performed for every received symbol.

Depending on how the ACS operations are scheduled, Viterbi decoder architectures can be roughly divided into state-parallel and state-serial. In state-parallel architectures, there is an ACS block for every state and all path metrics are computed simultaneously, which makes this architecture suitable for very high data rates. The resulting area is very large, growing exponentially with the constraint K. At the other end of the spectrum, state-serial architectures compute the path metrics sequentially using a constant number of ACS blocks, usually one or two (butterfly). However, the execution time grows exponentially with K. Thus, they are suitable only for medium-to-low data rates.

For each received symbol, all path metrics are updated and the 2^{K-1} decision bits are stored in the trace-back memory. As soon as a given number of symbols have been received, we start from the best current state and go back the trellis on the most likely path, producing one output bit for every state traversed. This operation is referred to as trace-back and reconstructs the original bit stream, albeit in reverse order. The longer the trace-back, the higher the likelihood that a bit is decoded correctly. By increasing the trace-back depth, the bit-error-rate asymptotically reaches a lower limit. **Figure 6.8** shows the dependence of BER on channel noise and trace-back depth.

In practical implementations, trace-back depth is limited by the available hardware and by the decoding latency. Unlike wireless LAN, latency is not an issue in digital broadcasting. Moreover, since broadcast data is usually organized as a continuous stream, trellis termination circuitry is not needed. These considerations relax the requirements imposed on the trace-back unit and allow for very efficient implementations. In our sequential design, a trace-back is performed every T received symbols, each trace-back operation producing T decoded bits.

6.1.1. The Trace-Back Unit

As part of a Viterbi decoder, the trace-back unit stores the decision bits (or survivors) computed by the ACS unit for every received symbol and generates the decoded output by performing a trace-back cycle on a predefined number of steps. The number of steps used to perform the

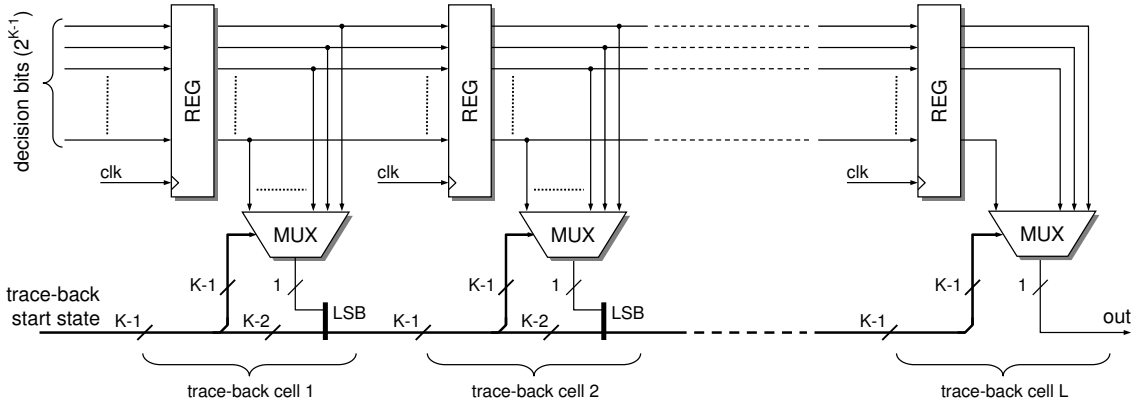


Figure 6.5.: Schematic of a parallel trace-back decoding window

trace-back is also referred to as the trace-back window length and will be denoted by L in the following.

Knowing the current state S_n and the decision bit associated with it, b_{S_n}, the previous state S_{n-1} can be found using the following formula:

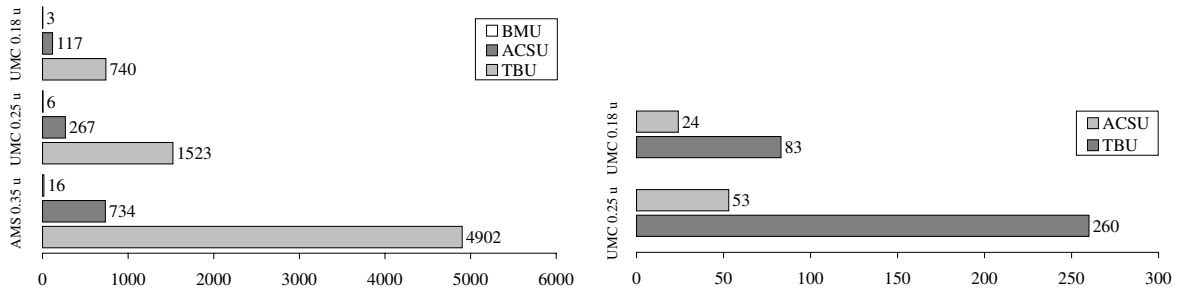
$$S_{n-1} = \frac{S_n}{2} + b_{S_n} \cdot 2^{K-2} \tag{6.1}$$

From an implementation point of view, the previous state is obtained by shifting the current state to the right and appending the decision bit as the MSB, requiring therefore no hardware resources. However, a multiplexer is needed to select the decision bit associated with the current state from the set of 2^{K-1} decision bits generated by the ACS unit.

A solution for the trace-back unit is presented in **Figure 6.5**, based on an idea found in [90], which implements directly (**6.1**). The shift register array is needed to store the decision bits for a number of L previous cycles. The most area will be taken by the multiplexors, whose size increases very fast with the constraint length:

$$SIZE_{MUX} = O(K \cdot 2^K) \tag{6.2}$$

Since a full length trace-back operation is performed every clock cycle on a purely combinational path, the operating frequency is seriously limited, even for low values of L. As there is no feedback loop, the circuit can be pipelined, by grouping the trace-back cells into stages and inserting register in between. For details about the optimum sizing of the pipeline, refer to our previous paper [121]



(a) Area (10^3 µm^2) (b) Power consumption @ 54MHz (mW)

Figure 6.6.: Standard cells synthesis results for the parallel Viterbi decoder

6.2. State-Parallel Architecture with Adjustable Trace-Back Unit

6.2.1. Area and Power Consumption of the Trace-Back Unit

In order to have a realistic image of the power consumption and area required by the trace-back unit compared to the rest of the Viterbi decoder, we have developed a fully parameterizable VHDL model and generated a specialized implementation for the IEEE 802.11a wireless LAN standard, with K = 7 and a pipelined trace-back unit with 16 stages and 5 cells per stage. It operates at 54 MHz, corresponding to the highest data rate specified by the standard (54 Mbps).

The VHDL description has been synthesized using three standard cell CMOS libraries, with feature sizes of 0.35, 0.25, and 0.18 µm respectively. The area and the power consumption are reported in **Figure 6.6**. Power consumption is not reported for the 0.35 µm library because the power characterization provided by the manufacturer was not accurate enough.

The results show clearly that the most area and highest percentage of the power consumption in a Viterbi decoder goes to the trace-back unit. In the following section we propose an architecture that allows the trace-back length to be adjusted dynamically, saving power by disabling the unused stages.

6.2.2. Adjustable Trace-Back Architecture

By analyzing **Figure 6.7**, it can be seen that if we need a lower decoding length L, all we have to do is to take the output from an earlier trace-back stage. Some stages will therefore remain unused and can be disabled in order to save power.

One solution is to use the enable signal for the registers to turn off any switching activity on the data path. A better one consists in applying clock gating to those stages we want to disable. This will reduce the power consumption to negligible levels, given only by the leakage current.

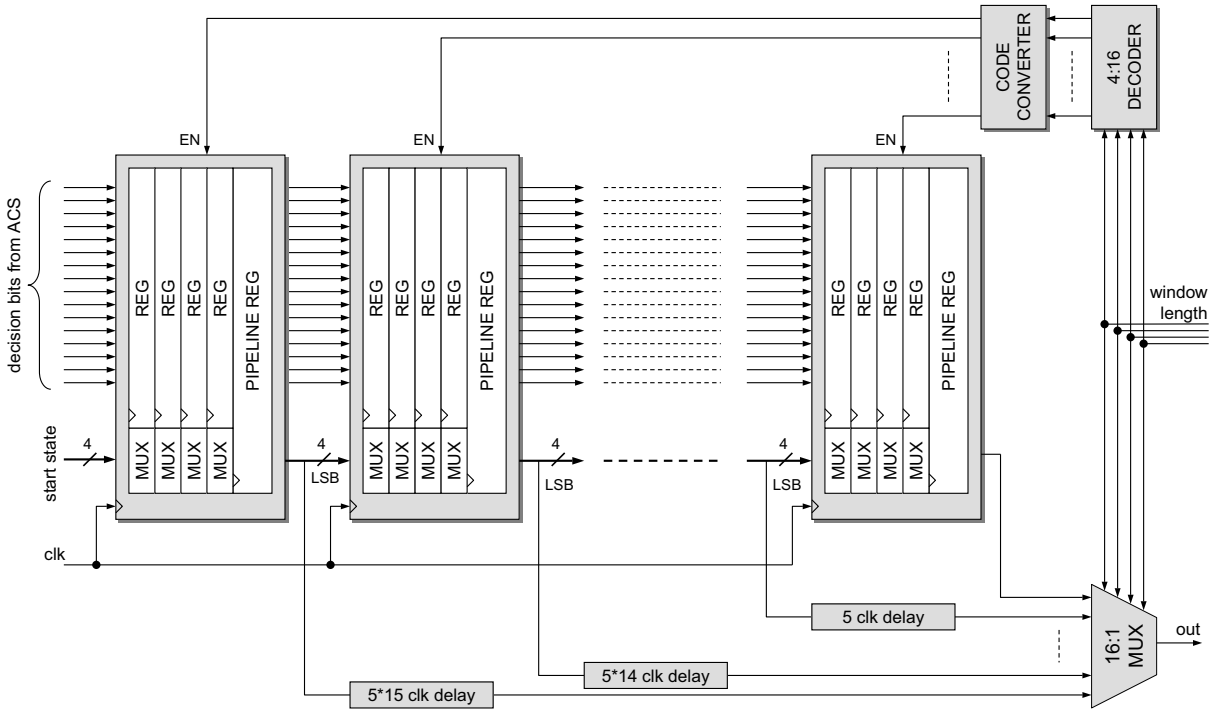


Figure 6.7.: Schematic of the proposed adjustable trace-back unit

The schematic of the proposed architecture is shown in **Figure 6.5** and allows using any of the above techniques. Each stage in the pipeline can be turned on and off using a generic *enable* signal.

The delay lines at the output of each stage prevent the loss of data when switching between different window lengths and ensure a constant latency regardless of the selected length. Moreover, only one delay line can be enabled at a time, in order to further reduce the power consumption.

The number of pipeline stages depends on the desired granularity for the adjustment. Let the number of stages be M and the number of cells in a stage P. In our example implementation, $P = 5$ and $M = 16$.

The controller generates the signals for enabling the trace-back stages (TBSEN) and the delay lines (DLEN), as well as for selecting the correct output through a multiplexer (OUTS). The sequence of these control signals depends on the direction of the adjustment and is shown below for both cases. IDX is the index of the length, with values between $1 \ldots M$.

When IDX is increased ($IDX : i \rightarrow j$, $i < j$):

- enable $TBS_{i+1} \ldots TBS_j$ ($TBSEN \leftarrow j$) and also DL_j ($DLEN_j \leftarrow 1$)
- wait for $D(M - j)$ clock cycles, so that valid data appears at the output of DL_j

• switch output multiplexer from DL_i to DL_j and disable DL_i ($DLEN_i \leftarrow 0$)

When IDX is decreased ($IDX : i \rightarrow j$, $i > j$):

• enable DL_j

• wait for $D(M - j)$ clock cycles, so that valid data appears at the output of DL_j

• switch output multiplexer from DL_i to DL_j, disable $TBS_{i+1} \ldots TBS_j$ ($TBSEN \leftarrow j$) and also DL_i ($DLEN_i \leftarrow 0$)

Simulations have shown that the smallest practically usable trace-back length for our implementation is 20, corresponding to 4 pipeline stages. For this case, the power consumption is 1/4 of the power for the full-size window (16 stages), corresponding to a power reduction of 75% for the trace-back window. Post synthesis simulations show that the controller accounts for less than 1% of the power and can be neglected. This results in an overall power reduction of 58% for the UMC 0.18 µm and 62% for the 0.25 µtechnology.

6.2.3. Finding the Optimum Window Length

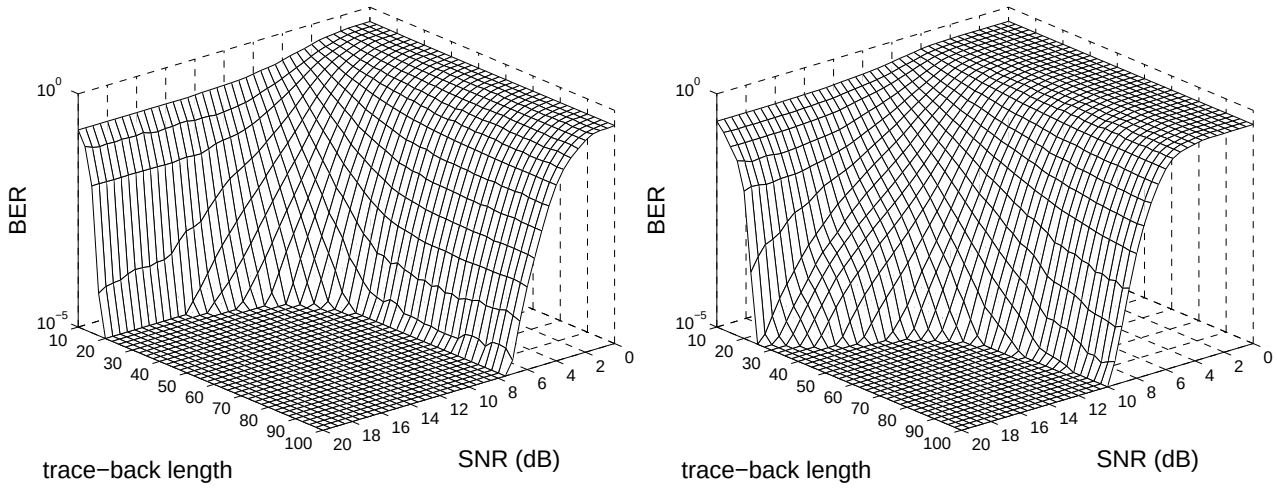
The goal here consists in finding the smallest window length that ensures a target bit error rate (BER) of the decoded output, for an estimated value of the SNR of the channel and a certain puncturing pattern. The process consists of two distinct steps.

The first step is performed only once, when a new decoder is designed. This involves building a look-up table with the optimum values for the window length, which has two entries: the desired BER of the decoded stream and the SNR of the channel. One such look-up table is built for every puncturing pattern. The IEEE 802.11a for instance, specifies a unpunctured mother code of rate 1/2 and two punctured codes of rate 2/3 and 3/4 respectively.

Extensive fixed-point simulations were performed on an exact model of the decoder, in order to determine the BER as a function of the SNR and the window length. For our investigation, we analyzed and simulated the decoder for the IEEE 802.11a standard [38] using a model and a simulation environment written in SystemC. The essential advantage of SystemC over VHDL or Verilog is a much higher simulation speed and flexibility. The results are presented in **Figure 6.8**. For details about the simulations refer to our paper [121]. I still need to add the contribution from this paper.

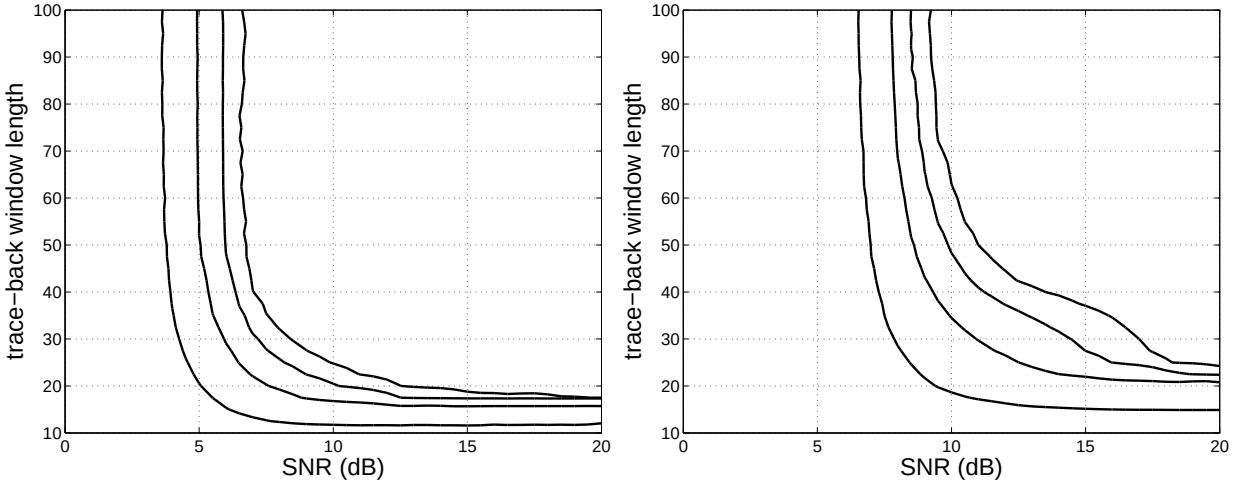
Based on these results, we obtain curves of constant BER. We have chosen four possible values for the target BER, which cover a wide range of requirements: 10^{-2}, 10^{-3}, 10^{-4} and 10^{-5}. The resulting curves are shown in **Figure 6.9**.

The curves of constant BER are discretized for the available window lengths (5, 10, 15, etc.), thus obtaining the optimum length for an estimated range of SNR. These discretized values are



(a) no puncturing (b) puncturing p1

Figure 6.8.: BER as a function of window length and channel SNR



(a) no puncturing (b) puncturing p1

Figure 6.9.: Lines of constant BER

then used to build the look-up tables.

The run-time part of the process consists in obtaining an estimate of the channel SNR, which is then used in conjunction with the target value for the BER and the current puncturing pattern to find the optimum value for the window length in the look-up table. Updating the window length to reflect changes in SNR or performance requirements is performed periodically.

Since estimating the SNR is a lengthy process, the look-up table and the look-up process can be implemented on an external processor. As soon as an updated SNR estimate becomes available, a new optimum value will be retrieved from the table at written to the trace-back controller. Since the process has no feedback, no stability problems occur.

6.3. Low-Area State-Serial Architecture

Our main goal was to minimize the required hardware resources, while applying power saving techniques where possible. We propose optimizations for all the blocks of a Viterbi decoder: branch metrics unit (BMU), add-compare-select unit (ACSU), and the trace-back unit (TBU). The architecture is mainly targeted for FPGA's, in the light of the new features offered by modern devices, such as embedded RAM and shift registers. Extremely low-area implementations are also possible for ASIC's, provided that the foundry offers a memory compiler that supports small RAM's.

As a representative example, we have chosen the low-cost Spartan-3 FPGA family from Xilinx. Spartan-3 devices feature between 4 and 96 true dual-port RAM blocks of 16Kbit each, which can be configured for different aspect ratios data widths ranging between 1 and 32 (power-of-2 only). Their operation is fully synchronous, with independent clocks for each port. Besides, the two LUT's in a slice can be configured as addressable 16-bit shift registers (with selectable output). Depending on the device, between 1536 and 66560 LUT's are available.

The dedicated FPGA resources can be used in VHDL by hand-instantiating vendor-specific primitives or by direct inference from VHDL constructs. Only the latter solution allows for fully generic and platform-independent designs. The VHDL design of our proposed architecture is completely generic, all parameters being definable upon the instantiation of the core, without the need of a user package. Special attention has been paid to minimizing the number of RAM blocks, as they are a scarce resource in FPGA.

Branch Metrics Unit. As it can be seen in **Figure 6.4**, the straightforward implementation of the BMU requires a significant number of adders in order to compute every possible branch metric. For the convolutional code specified by DAB and DRM, 16 branch metrics must be computed in parallel, which requires 48 adders. The proposed BMU architecture, shown in **Figure 6.10**, computes the metrics sequentially using three adders instead. The appropriate soft-bits are selected through multiplexors controlled by a counter which counts from 0 to 15. As they are computed, the branch metrics are shifted into a shift register of length 16 to

accommodate all metrics, from where they are selected by ACSU through the output MUX. The SHREG/MUX combination is ideally implemented using the addressable 16-bit shift registers available in Spartan-3 FPGA's. These optimizations resulted in 70% reduction of the FPGA slice count.

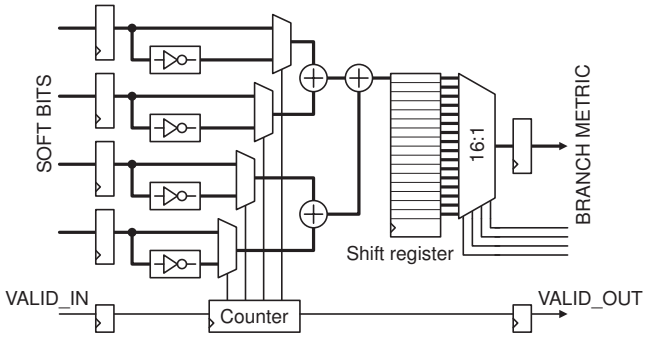


Figure 6.10.: Sequential BMU architecture

Add-Compare-Select Unit. The proposed architecture uses synchronous dual-port memory for metrics storage. In order to save memory, an in-place metric update scheme is employed. The in-place algorithm requires that for each ACS butterfly, the two read addresses and the two write addresses be the same. The natural trellis allocation in **Figure 6.2** does not meet this requirement. A modified version with reordered state addresses is shown in **Figure 6.11**.

One can see that the allocation of the metrics changes from stage to stage, the allocation pattern repeating itself after $K-1$ stages. It is worth observing that the resulting trellis resembles an FFT graph, so that the insights from the in-place computation of the FFT [12] can be applied. **Table 6.1** shows the butterfly allocation for an eight-state decoder as an example. Metric addresses can be easily obtained by rotating the state counter with the stage index [78].

Since only one memory bank is used for storing the metrics, the two path metrics for each ACS operation are read sequentially, the first being stored in a temporary register. Moreover, because each pair of path metrics is used to compute two new path metrics, extra registers are necessary to keep both old path metrics unchanged for two clock cycles. In the first cycle

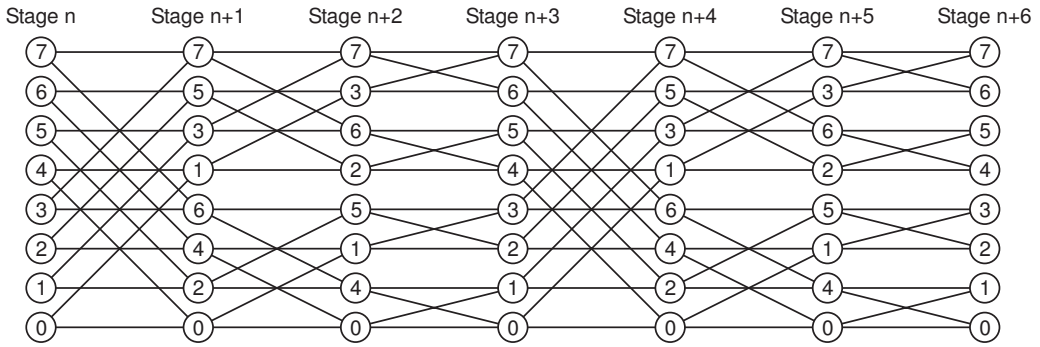


Figure 6.11.: State transition graph for in-place addressing

Butterfly	Stage		
$b_1\ b_0$	0	1	2
Low Addr	$\rightarrow$ 0 b_1 b_0	b_0 0 b_1	b_1 b_0 0
High Addr	$\rightarrow$ 1 b_1 b_0	b_0 1 b_1	b_1 b_0 1
0 0	(0;4)	(0;2)	(0;1)
0 1	(1;5)	(4;6)	(2;3)
1 0	(2;6)	(1;3)	(4;5)
1 1	(3;7)	(5;7)	(6;7)

Table 6.1.: Memory addressing for in-place metric computation

the ACS block produces the metric for the even state while in the second the metric for the odd state. The branch metrics are selected from BMU based on the output produced by the convolutional encoder for the transitions from the two candidate states, assuming a 0 at the input (even current state). For odd current states, the branch metrics need only be swapped, as the encoder outputs are the same (see **Figure 6.3**). The schematic of the proposed sequential ACS architecture is shown in **Figure 6.12**.

The memory read, the temporary registers, ACS, and the write-back form a five-cycle pipeline. That is why the write address is a delayed version of the read address. In order to save power, the data path and the memory are disabled when the ACS unit is not active. The schematic of the control unit is shown in **Figure 6.13**, where the state and the stage counters can be seen, together with the barrel shifter for computing the memory address. An optimization we propose is to implement the stage counter as one-hot, which results in a simplified and faster shifter. Besides the address, the current state also needs to be delayed since it is needed for iteratively determining of the state with the best path metrics (not shown). Also part of the ACS unit, is the circuit (not shown) for detecting the normalization condition when all path metrics exceed a predefined threshold.

Trace-Back Unit. For every received symbol, a set of 64 decision bits from the ACS unit are saved in the trace-back memory. This would require a 64-bit wide memory, with a size equal to the trace-back depth. In this case, two RAM blocks would have to be concatenated, even for small trace-back depths, which leads to an inefficient implementation for FPGA, where RAM blocks are maximum 32-bit wide. Our solution consists in splitting the 64-bit decision vector into shorter segments, e.g. 16-bit. This enables the use a memory with a shorter data bus, and thus implementable with only one RAM block on FPGA. An additional benefit is the shortening of the decision-bit shift register in the ACS unit (**Figure 6.12**). The amount of splitting is a generic parameter of the design.

The segment index is given directly by the MSBs of the state counter (**Figure 6.13**). At the same time, it is used as the LSBs of the trace-back memory write address since the decision-bit segments are stored at adjacent addresses in memory. The trace-back memory is used as a

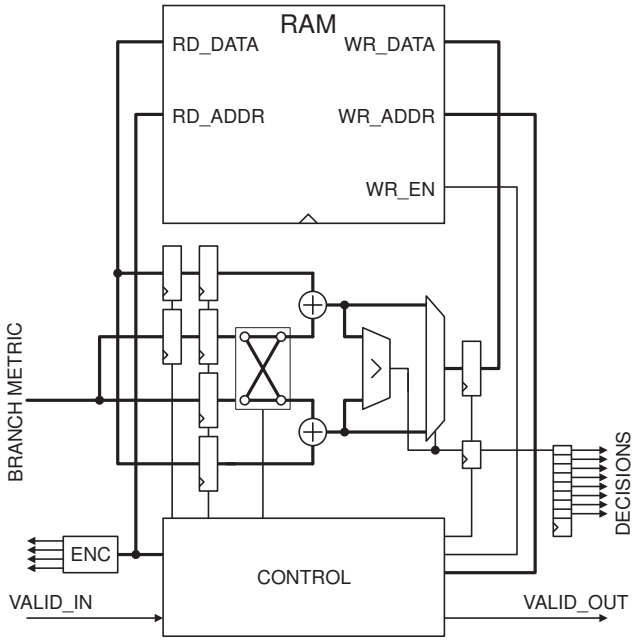


Figure 6.12.: Proposed sequential ACS architecture

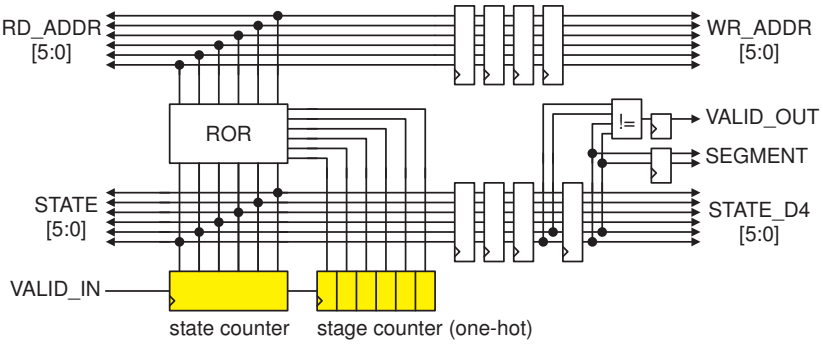


Figure 6.13.: Control unit for the sequential ACS

Parameter	Description
NPOLYS	Number of generator polynomials
CONSTR	Constraint length
POLYS	Generator polynomials concatenated in a vector
SBW	Soft-bit width
BMW	Branch-metric width
PMW	Path-metric width
DVRF	Decision vector reduction factor = 2^{DVRF}
TBSZ	Trace-back memory size = 2^{TBSZ}
NDB	Number of decoded bits in one TB = 2^{NDB}

Table 6.2.: Generic parameters of the Viterbi decoder design

circular buffer with two running pointers. The write-pointer is incremented after a full set of decision bits has been stored, whereas the read-pointer is initialized with the value of the write-pointer at the beginning of a trace-back operation, then counts down through the entire memory to access past decision bits.

Another important parameter of the trace-back unit is the number of bits decoded in a trace-back operation, which also gives the interval between two successive trace-backs. The longer the interval, the more power can be saved, due to reduced memory activity. The problem of obtaining the decoded bits in reverse order is solved by shifting them in a shift register, then reading them in parallel. During trace-back, not all addresses are read, but only the needed segment, which further contributes to reducing power.

6.3.1. Implementation Results

The proposed architecture has been modeled in VHDL at register-transfer level. The resulting design is completely generic and technology independent, all parameters of the Viterbi decoder being definable when the core is instantiated. **Table 6.2** shows these generic parameters. The first group define the convolutional code, while the others are implementation-related parameters.

In order to demonstrate the suitability of the proposed architecture for FPGA implementation, we have considered the Viterbi decoder for the DAB and DRM standards, which specify a convolutional code with 4 generator polynomials of constraint 7. As an example we have chosen the Spartan-3 FPGA family and the XST synthesis tool from Xilinx. The implementation parameters have been assigned the following realistic values: SBW=3, BMW=5, PMW=8, DVRF=2, TBSZ=6, and NDB=3. Thus, the branch metrics memory will be 64×8-bit and the trace-back memory 256×16-bit. The results are shown in **Table 6.3**, where the last column represents the number of logic elements available in the smallest Spartan-3 device, XC3s50.

	BMU	ACSU	TBU	Top	Available
Slices	25	91	27	128	768
Slice FF's	11	131	34	176	1536
LUT's	40	93	40	164	1536
Block RAM's	0	1	1	2	4
Max. f_{clk}	161 MHz	153 MHz	183 MHz	153 MHz	

Table 6.3.: Synthesis results for Xilinx Spartan-3 FPGA

	State-serial			State-parallel		
	BMU	ACSU	TBU	BMU	ACSU	TBU
Slices	25	91	27	88	1506	3354
Slice FFs	11	131	34	82	659	4204
LUTs	40	93	40	132	2456	1948

Table 6.4.: Comparison between state-serial and state-parallel implementations

Except for the two RAM blocks, only 16% of the slices are utilized, 50% thereof being used by the ACS unit alone. The figures are as reported by the synthesizer before place & route.

The throughput is limited by the maximum clock frequency and by the constraint length. At 153 MHz and K=7, the throughput is 2.39 Mbps, which exceeds the specifications of the DAB standard (max. 1728 kbps for data services and max. 384 kbps for audio streams). The lowest clock frequency that satisfies these requirements is approx. 111 MHz, which is easily achieved even for slower devices. The DRM specifications are also met since DRM offers lower data rates than DAB.

In order to show the difference in hardware resources between the state-serial and the state-parallel approach, we have compared the state-serial decoder with a well designed state-parallel one for the same parameters, except for DVRF and NDB which are not applicable in this case. The comparative results in terms of FPGA resources are shown in **Table 6.4**, where a significant difference can be observed. Especially inefficient for FPGA is the trace-back unit, since it is implemented using multiple shift-registers in parallel.

The proposed state-serial Viterbi decoder architecture requires a minimum of hardware resources, targeted at digital radio applications. The architecture is especially suited for FPGA implementations since both the ACS and the trace-back unit take advantage of the embedded RAM blocks that have become standard in FPGA devices. Using DAB and DRM as examples, we have shown that an FPGA implementation can meet the throughput requirements for modern digital radio standards. However, the proposed architecture is also suitable for other communication applications with medium-to-low data rates.

By combining the flexibility of DSP's with the performance of ASIC's, FPGA's are becoming a serious alternative for consumer applications. The architecture proposed here, together with similar results we obtained for an area-optimal FFT core (**Section 7.2**), show that a complete digital radio can be implemented in a single low-cost FPGA. The key strategies are to employ area-efficient sequential architectures and to exploit the features offered by modern FPGA's, such as RAM, multipliers and shift registers.