Design Space Exploration of Hard-Decision Viterbi Decoding: Algorithm and VLSI Implementation

Irfan Habib, Özgün Paker, Member, IEEE, and Sergei Sawitzki, Member, IEEE

Abstract—Viterbi algorithm is widely used as a decoding technique for convolutional codes as well as a bit detection method in storage devices. The design space for VLSI implementation of Viterbi decoders is huge, involving choices of throughput, latency, area, and power. Even for a fixed set of parameters like constraint length, encoder polynomials and trace-back depth, the task of designing a Viterbi decoder is quite complex and requires significant effort. Sometimes, due to incomplete design space exploration or incorrect analysis, a suboptimal design is chosen. This work analyzes the design complexity by applying most of the known VLSI implementation techniques for hard-decision Viterbi decoding to a different set of code parameters. The conclusions are based on real designs for which actual synthesis and layouts were obtained. In authors’ view, due to the depth covered, it is the most comprehensive analysis of the topic published so far.

Index Terms—Viterbi algorithm, VLSI design, survey.

I. INTRODUCTION

VITERBI decoder is one of the most widely used components in digital communications and storage devices. Although its VLSI implementation is studied in depth over the last 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 the advancement in CMOS 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], [8], and [9]. Still, in authors’ view, a systematic and comprehensive analysis summarizing and characterizing as many of the trade-offs and implementation techniques as possible is missing. This contribution presents such a survey, providing designers with clear guidelines and references to find the best solution for every specific case.

The scope of the paper is limited to the “classical” Viterbi algorithm. Trellis decoding techniques like MAP-decoding, T- and M- algorithms are not discussed in detail. In addition, almost all of the studied designs optimizations, although proven by synthesis for a specific target CMOS technology, are technology independent, so the conclusions should remain valid for the next two to three solid-state device generations. For this reason, transistor and interconnect level improvements are not considered in this paper.

We assume that the reader is familiar with convolutional codes and basic implementation techniques of the Viterbi algorithm, so we limit the discussion of these subjects in the following sections to a minimum. A good summary of references to both the topics can be found in [8] and [10].

The rest of this paper is organized as follows. Section II gives a brief description of convolutional codes and the Viterbi algorithm and introduces the parameters influencing the design complexity. Sections III–V discuss design decisions for every subunit of the decoder. Finally, Section VI draws some conclusions.

All area and timing results presented in this paper are based on the standard cell 90-nm CMOS technology.

II. GENERAL STRUCTURE OF THE VITERBI ALGORITHM

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. Generally speaking, every input bit is encoded using n output bits, so the coding rate is defined as 1/n (or k/n if k input bits are used). The constraint length of the code K is defined as the length of the shift register plus one. Generally, generator polynomials define, which bits in the input stream have to be added to form the output. An encoder is completely described by n polynomials of degree K 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. Note that for the state encoding, B_{k-1} represents the least significant bit and B_{k-2} the most significant bit respectively. A transition in the trellis diagram is called a branch. For a binary convolutional code, every state in the trellis has two incoming branches representing the transmission of one and zero, respectively.

Manuscript received July 30, 2008; revised December 04, 2008.

I. Habib is with the Corporate I&T, NXP Semiconductors, 5656AE Eindhoven, The Netherlands (e-mail: irfan.habib@nxp.com).

Ö. Paker is with the Advanced R&D, ST-Ericsson, 5656AE Eindhoven, The Netherlands (e-mail: ozgun.paker@stericsson.com).

S. Sawitzki is with the University of Applied Sciences FH Wedel, 22880 Wedel, Germany (e-mail: sergei.sawitzki@gmail.com).

Digital Object Identifier 10.1109/TVLSI.2009.2017024
Every state transition in the encoder results in one codeword being produced. After the transmission over a noisy channel, the codewords 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 codewords 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 branch metrics are added to the previously stored path metrics 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.

The PMU 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 (or less) clock cycles. From this point on, the decoded output sequence can be reconstructed.

Coding rate $1/n$, constraint length $K$, traceback depth TBD, 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.

III. BRANCH METRIC UNIT

A. Design Space

BMU is typically the smallest unit of Viterbi decoder. Its complexity increases exponentially with $n$ (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. 3 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.

B. Branch Metric Precision

BM should be able to hold the maximum possible difference (distance) between ideal and received symbols. For hard input (softbits = 1), 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). In such case, the maximum distance is equal to the symbol length $n$. So, BM precision is given by

$$BM_{precision} = \lceil \log_2 n \rceil + 1.$$ (1)
In soft input decoding, each received symbol is represented by more than one bit (softbits > 1). In this case, the maximum possible branch metric is calculated as distance

$$BM_{BW} = \left\lceil \log_2(2^{\text{softbits} - 1}) \times n \right\rceil + 1.$$  \hspace{1cm} (2)

C. Radix-2 BMU

The hardware cost of BMU from Fig. 3 in terms of areas of basic components can be expressed as follows:

$$BMU_{2\text{area}} = 2^n \times (HA_{\text{area}} + FA_{\text{area}} \times (\text{softbits} - 1)) + \text{softbits} \times n \times \text{INV_{area}}$$ \hspace{1cm} (3)

where $HA_{\text{area}}$, $FA_{\text{area}}$, $\text{INV_{area}}$ denote the area of a half adder, full adder, and inverter, respectively.

D. Radix-4 BMU

An implementation of radix-4 BMU (processing two code-words per clock cycle) is shown in Fig. 4. The underlying radix-2 BMs are added to form radix-4 BMs [11]. For Radix-4 BMU, the area cost is expressed as follows:

$$BMU_{4\text{area}} = 2^{2\times n} \times (HA_{\text{area}} + FA_{\text{area}} \times \text{softbits}) + 2 \times 2^n \times (HA_{\text{area}} + FA_{\text{area}} \times (\text{softbits} - 1)) + n \times \text{softbits} \times \text{INV_{area}}.$$ \hspace{1cm} (4)

BMUs are generated assuming that every possible combination of $n$ bits is a valid encoder output. But as value $n$ 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.

IV. PATH METRIC UNIT

A. Design Space

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 as Fig. 5 shows).

Optimization techniques of PMU in Viterbi decoder implementation are shown in Fig. 6. 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 a single recursion (this is called look ahead). If look ahead is short then area penalty is not high. Radix-4 lookahead (i.e., processing two bits at a time) is a commonly used technique to increase decoder’s throughput.
Bit level techniques optimize the add-compare-select operations in PM computation and decision process. Bit level optimizations can be used to speed up the ACS operation itself. These techniques differ from word level techniques by having a lower area penalty. Fig. 7 shows the various bit level optimization techniques. Compare-select-add (CSA) is a retimed variation of the ACS operation as shown in Fig. 8. It computes all possible paths originating from a node and based on decision it retains the minimum PM path. Thus, addition and comparison can be processed in parallel.

Further, addition and comparison operation can be implemented using different arithmetics. Carry-look-ahead (CLA) technique [23] uses carry look ahead adders for addition and comparison (subtraction) operation. Ripple-add carry-chain compare exploits bit level parallelism between addition and comparison operations. In this technique, subtraction is used for comparison. Both addition and subtraction operations start with the least significant bit (LSB). Thus, as soon as LSB of the sum is computed, it can be used for comparison (see Fig. 9). In the carry save approach, instead of propagating the carry from LSB to the most significant bit (MSB), carry generated during each bit addition is saved. The comparison starts with MSB, taking into account the saved carry bits. This makes the comparison operation quite complex. The advantage, however, is that the addition and comparison operations can be pipelined at bit level to reduce the critical path. Due to the complex compare operation there is a significant area penalty.

In the bit serial approach the addition and comparison operations are carried out in serial manner. It uses 1 bit adder over $N$ cycles to compute $N$ bit addition (or subtraction). Thus area saving in addition and comparison operations is achieved.

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.

The area and throughput comparisons of system level techniques are presented in Table I. Throughput and complexity are normalized to radix-2, fully parallel (1 ACS/state) Viterbi decoder $K$ denotes constraint length while $M$ denotes look-ahead steps. Number of ACS units is denoted by $P$.

In this paper only bit level optimizations are discussed in depth, since analysis of the algorithmic and word level techniques concerning their effects on area and throughput is rather straightforward as the summary in Table I suggests.

### B. Path Metric Precision

The register temporarily storing path metrics should be wide enough to avoid overflow errors in the PMU operations. Modulo arithmetic is usually used for this purpose [29]. PM bitwidth is determined as follows:

$$ PM_{BW} \geq \left\lceil \log_2 B + \log_2 (2 \times 2 \times (K - 1)) \right\rceil $$

where $B$ is given as

$$ B = (2^{\text{soft bits}} - 1) \times n. $$
C. Choice Among Various PMU Implementations

As mentioned before, there are several options to implement the ACS operation. Radix-2 ACS (see Fig. 5) is a basic design with lowest area and throughput figures. Other techniques such as CSA (see Fig. 8) and Radix-4 (see Fig. 10) provide a higher throughput than radix-2 ACS at the cost of higher area as will be discussed in this section later. Note that the architecture of the adders and subtracters for the ACS implementation is an orthogonal design choice. More specifically, the addition and subtraction operations can be described either in a behavioral fashion (i.e., giving the freedom to choose a particular implementation to the synthesis tool) or by instantiating specific arithmetic circuits manually such as ripple carry or carry look ahead (CLA) adders and subtracters.

As PMU is a fairly regular unit where a basic module (ACS) is instantiated several times (e.g., $2^{K-1}$ in case of the fully parallel approach), it is especially important to optimize this module. For $N$ bit addition, the delay of CLA implemented in a logarithmic configuration [23] can be expressed as

$$\text{CLA}_{\text{delay}} = T_{\text{XOR}} + (T_{\text{AND}} + T_{\text{OR}}) \times \log_2 N + T_{\text{FA}} \quad (7)$$

where delays of logic components such as XOR, AND, OR, and full adder are denoted as $T_{\text{XOR}}, T_{\text{AND}}, T_{\text{OR}},$ and $T_{\text{FA}}$ respectively. Similarly, delay through ripple carry adder can be expressed as a function of half-adder and full adder delays

$$\text{RCA}_{\text{delay}} = T_{\text{HA}} + (N - 1) \times T_{\text{FA}}. \quad (8)$$

As indicated in [23] the efficiency of CLA is better than ripple only for large values of $N$, e.g., $N > 8$.

Fig. 5 highlights the critical path in radix-2 ACS. When implemented in CLA configuration, each circle represents an addition/subtraction with precision $PM_{\text{BW}}$. Assuming no parallelism between addition and comparison operations, critical path delay can be expressed as

$$\text{CLA}_{\text{ACS}}_{\text{delay}} = 2 \times \text{CLA}_{\text{delay}} + T_{\text{MUX}}$$

$$= 2 \times (T_{\text{XOR}} + (T_{\text{AND}} + T_{\text{OR}}) \times \log_2 PM_{\text{BW}} + T_{\text{FA}}) + T_{\text{MUX}}. \quad (9)$$

For ACS based on ripple-carry adders, critical path delay can be expressed as

$$\text{RCA}_{\text{ACS}}_{\text{delay}} = (PM_{\text{BW}} + 1) \times T_{\text{FA}} + T_{\text{MUX}}. \quad (10)$$

With CLA implementation, we can observe a logarithmic increase in critical path delay with path metric precision rather than a linear one as in ripple implementation. As mentioned...
earlier, by using ripple configuration in radix-2 ACS, bit level parallelism between addition and comparison can be exploited (see Fig. 9). Therefore, the delay of addition and comparison operations is just one full adder delay more than the $P_{MBW}$ bits ripple carry addition. While using CLA for addition operation in radix-2 ACS the parallelism between addition and comparison operations can not be fully exploited. Thus, addition and comparison operations require nearly twice the delay of a $P_{MBW}$ bits CLA addition. Therefore, using CLA for addition in radix-2 ACS for low $P_{MBW}$ values will not improve the critical path delay significantly. The advantage of using CLA becomes apparent when the number of precision bits increases. The critical path delay for CLA is $\approx (2 \times \log_2 P_{MBW})$ while for ripple carry, it is $\approx P_{MBW}$.

Fig. 8 highlights the critical path of a CSA implementation. If the adders of Fig. 8 are implemented using the CLA technique, delay can be expressed as

$$CLA_{CSA}_{\text{delay}} = CLA_{\text{delay}} + T_{MUX}$$
$$= (T_{XOR} + (T_{AND} + T_{OR})$$
$$\times \log_2 P_{MBW} + T_{FA}) + T_{MUX}. \quad (11)$$

While in case of a ripple carry implementation for the same adders, the delay can be expressed as

$$RCA_{CSA}_{\text{delay}} = P_{MBW} \times T_{FA} + T_{MUX}. \quad (12)$$

In case of CSA, the critical path delay using ripple carry adders is $\approx P_{MBW}$, while using CLA, it is $\approx \log_2 P_{MBW}$. Here the advantage of using CLA becomes apparent. Thus for higher values of $P_{MBW}$, CSA using CLA adders and subtracters is a better design choice to achieve high throughput.

Fig. 10 highlights the critical path through a radix-4 ACS. It can be observed that hierarchical comparisons cause major delay in this design. Alternative version compares each sum pair-wise and then encodes the results of the comparison, as shown in Fig. 11. Depending on the $P_{MBW}$ value, the critical path through the radix-4 ACS becomes shorter in this case. Encoder and MUX can also be combined to form an integrated selection unit. The delay of Radix-4 ACS while using CLA for addition and comparison operation can be expressed as

$$CLA_{R4}_{\text{delay}} = 2 \times CLA_{\text{delay}} + T_{encoder} + T_{MUX}$$
$$= 2 \times (T_{XOR} + (T_{AND} + T_{OR})$$
$$\times \log_2 P_{MBW} + T_{FA})$$
$$+ T_{encoder} + T_{MUX}. \quad (13)$$

The delay of Radix-4 ACS using ripple carry technique for addition operation can be expressed as

$$RCA_{R4}_{\text{delay}} = (P_{MBW} + 1)T_{FA} + T_{encoder} + T_{MUX}. \quad (14)$$

As radix-4 ACS is larger than radix-2 ACS or CSA, a variety of configurations are possible in its implementation. Area and throughput requirements determine the choice of a certain configuration. For radix-4, the critical path delay is $\approx 2 \times \log_2 P_{MBW}$ for CLA and $\approx P_{MBW}$ for ripple carry adder. As compared to the delay of radix-2 ACS, there is an overhead of an encoder. For the cascade comparison as shown in Fig. 10 the delay will be $\approx 3 \times \log_2 P_{MBW}$. For low values of $K$ and softbits, $P_{MBW}$ is low. Using CLA for lower number of bits does not provide any throughput benefit.

To compare the designs in terms of area and logic delay and to determine the optimal implementation of each design, radix-2 ACS and CSA as well as radix-4 ACS units were implemented and synthesized using the above mentioned techniques. Different implementations of the designs were synthesized with a relaxed timing constraint to be able to see the effect of the architectural choices. Figs. 12 and 13 show the plots of area and critical path delay of different PMUs as obtained from synthesis experiments. As expected, the conventional radix-2 ACS is a better choice in terms of area, provided that the throughput requirement is met. For instance, the gain in area for the CLA implementation can be up to 50%.

Advantage of the CLA becomes visible only for higher $P_{MBW}$ values. For instance, using carry-lookahead in the CSA technique, achieves 68% higher throughput at a cost of 62% higher area compared to a radix-2 ACS with ripple carry adders. In [7], it is mentioned that CSA can achieve twice as much throughput as ACS. However, from Fig. 13, it is clear that CSA does not have a critical path half that of ACS in any implementation. This difference can be explained by the fact that the data in [7] come from the prelayout analysis for a single ACS/CSA operation and fixed parameter set, whereby Fig. 13 is based on post-layout analysis of the complete PMU for many different parameter values. Contrary to the widespread opinion, we conclude that the throughput advantage of CSA over ACS is much less than factor 2 compared to almost doubled area cost.
bit addition (or subtraction). Considerable 5 and 6 are almost identical). Note that

As seen in the figure, there is a significant gain in BER when moving from hard-input to 4 bits per symbol, i.e., softbits and the number of bits to process per clock cycle (radix-2 versus radix-4). These parameters will be discussed in the following subsections in more detail.

1) Softbits: Number of bits per symbol, i.e., softbits influences not only the area and delay of the PMU but also the performance of the Viterbi decoder in terms of Bit Error Rate (BER). Fig. 14 shows the dependency of BER on the $E_b/N_0$ (energy per bit/noise) ratio for additive white Gaussian noise (AWGN) channel with $K = 3$. As seen in the figure, there is a significant gain in BER when moving from hard-input to 4 bits per symbol. Further increase has almost invisible gains (indeed, the curves for $\text{softbits} = 5$ and 6 are almost identical). Note that with channel models other than AWGN, increasing the input

### Table II

<table>
<thead>
<tr>
<th>$K$</th>
<th>BS</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = 4$</td>
<td>0.006</td>
<td>0.007</td>
</tr>
<tr>
<td>$K = 9$</td>
<td>1.92</td>
<td>1.36</td>
</tr>
<tr>
<td>Throughput (Mbps)</td>
<td>57.87</td>
<td>735</td>
</tr>
</tbody>
</table>

#### E. Bit-Serial (BS) ACS

A BS design for ACS has been proposed in [24] for a high constraint length, low throughput Viterbi decoder. This design uses some circuit level optimizations and custom designed FIFOs and aims to reduce power and area. The bit serial approach uses 1-bit adder over $P_{MBW}$ number of cycles to implement $P_{MBW}$ bit addition (or subtraction). Considerable area saving is expected if $P_{MBW}$ is large. Referring to (5) and (6), the $P_{MBW}$ value is high for high values of $K$ and softbits. To compare bit-serial approach with other designs, a corresponding architecture was implemented with a standard cell library. The area and delay comparisons are shown in Table II. Area reduction nearly by a factor of two is seen for BS for $K = 9$. In case of $K = 4$, the area advantage is negligible. BM precision and PM precision determine the number of 1-bit adders (area for addition operation) in bit-serial approach and consequently the area advantage in using bit-serial over bit-parallel. Thus, bit-serial approach has an advantage in cases where the throughput requirements are met while the constraint length is large.

Note that two different clock frequencies are required for a PMU built upon bit-serial ACS, since path metrics unit needs several clock cycles to process a single input symbol while other units typically run at one symbol per clock cycle rate.

#### F. Critical Factors in PMU Design

PMU contains a feedback loop. Therefore, in general, throughput cannot be increased by pipelining (though as already mentioned, some look-ahead techniques allow loop unrolling and pipelining, but are extremely expensive in terms of area and power consumption). Thus, PMU is a bottleneck in achieving high throughput for the Viterbi decoder. At the same time, it occupies a substantial area considering the whole design. The most critical design parameters for the PMU are the input bitwidth, i.e., softbits and the number of bits to process per clock cycle (radix-2 versus radix-4). These parameters will be discussed in the following subsections in more detail.

1) Softbits: Number of bits per symbol, i.e., softbits influences not only the area and delay of the PMU but also the performance of the Viterbi decoder in terms of Bit Error Rate (BER). Fig. 14 shows the dependency of BER on the $E_b/N_0$ (energy per bit/noise) ratio for additive white Gaussian noise (AWGN) channel with $K = 3$. As seen in the figure, there is a significant gain in BER when moving from hard-input to 4 bits per symbol. Further increase has almost invisible gains (indeed, the curves for $\text{softbits} = 5$ and 6 are almost identical). Note that with channel models other than AWGN, increasing the input

---

**Figure 12.** Area figures for various ACS implementation techniques ($K = 4$ works with softbits $= 2, P_{MBW} = 3$, and $P_{BM} = 7$; $K = 7$ works with softbits $= 6, P_{BM} = 7$, and $P_{BM} = 12$).

**Figure 13.** Critical path delay for various ACS implementation techniques ($K = 4$ works with softbits $= 2, P_{BM} = 3$, and $P_{BM} = 7$; $K = 7$ works with softbits $= 6, P_{BM} = 7$, and $P_{BM} = 12$).
bitwidth beyond 4 bits still may make sense. However, the designer should always keep in mind, that increasing the \textit{softbits} parameter causes an increase in branch metric and path metric precision (and area) and in most cases decreases the throughput as illustrated by Figs. 15–17 for radix-2 decoders with different \( K \) values (conventional ACS, behavioral description).

Fig. 15 shows the affect of input bitwidth on the critical path delay for various \( K \). As expected, in majority of the cases there is a linear increase in the critical path delay.

Fig. 16 displays the variation in area for the PMU with increasing \textit{softbits} for various \( K \). A clear linear increase in area is visible in all cases as well, since BM and PM precisions increase linearly with the input bitwidth. This is because the number of full adders, multiplexers, and registers grow at the same rate. Consequently, the graph in Fig. 16 is a nice illustration of what one would expect from basic theoretical analysis.

The increase in area for lower values of \( K \) is not very apparent in Fig. 16. That is why Fig. 17 was created to show the percentage increase in area over the area of PMU for hard-input decoder. It also shows that that there is a significant area penalty when increasing the \textit{softbits} for all values of \( K \). From (5) and (6), one can conclude that PM precision increases with the logarithm of \( K \). At the same time, it increases linearly with input bitwidth, thus the influence of softbits on PM precision is more pronounced for lower values of \( K \). This explains the highest relative increase in area for the PMU for \( K = 3 \).

2) Radix-2 versus Radix-4: Fig. 18 shows the throughput achievable by using radix-2 and radix-4 decoder implementations. These results are based on timing-driven synthesis to achieve the highest throughput figures possible. The resulting throughputs are calculated by the synthesis tool, taking into account all clock related issues like clock skew, uncertainty, insertion delay etc., before layout. To check the back-end affects, a number of layouts were produced as well. The results indicate that on average, the deviation between pre- and post-layout critical path delay is less than 5%.

When using the radix-4 technique, two output bits are produced per clock cycle. Therefore, a higher throughput can be achieved even if the decoder is running at a lower clock frequency. Clock frequency becomes critical, especially in deep sub micron range. Higher clock frequency leads to cross talk.
issues and higher power consumption besides decreasing signal reliability. That is why radix-4 is potentially a suitable choice in deep submicron domain, especially if high throughput (e.g., > 700 Mb/s) is targeted without pushing the edge on the operating clock frequency. However, especially for higher $K$ values, area penalty of radix-4 compared to radix-2 can become quite significant as Fig. 19 shows. Alternatively, radix-4 technique can be used to achieve higher throughput at a higher area cost.

Figs. 20 and 21 show the relative increase in area and throughput for radix-4 PMU compared to radix-2 PMU for different values of $K$. Since the logic of radix-4 ACS is more complex compared to radix-2 ACS, the maximum clock frequency it can run at is lower. This implies that in case of an optimization towards highest achievable throughput, a radix-4 ACS is not exactly a factor of 2 faster than a radix-2 ACS as one would expect in the case of synthesis under relaxed timing constraints. Still depending on constraint length, throughput can be increased by 40–70% (see Fig. 21) provided that the PMU area penalty of a factor 3.4–4.2 (see Fig. 20) can be afforded. Especially when radix-2 designs cannot meet the throughput requirements, this factor is a reasonable price to pay. Note that for $K = 3$, comparable results were reported in [7]. Since the PMU is only one part of the Viterbi decoder, the overall penalty paid in terms of silicon area should be scaled accordingly (the area breakdown of a whole Viterbi decoder for $K = 7$ will be presented below).

Figs. 22–25 summarize throughput and area tradeoffs in the radix-2 and radix-4 PMU design for different constraint lengths and input bitwidths.

To prove our findings on the PMU design, radix-2 and radix-4 Viterbi decoders for $K = 7$, soft bits = 3, and TBD = 128 were implemented. The trace-back depth selection was done based on the simulations for a wireless fading channel under very noisy conditions (the value results from a baseband design project carried out by one of the authors). As known from literature, in case of AWGN channel, lower values for TBD, e.g., 5–6 K, would suffice. The synthesis was ran for worst-case military conditions. The post-layout netlists were simulated with timing to determine the maximum possible throughput. Two cases referenced as constrained and relaxed synthesis were considered to get more objective figures. Constrained synthesis runs
TABLE III

<table>
<thead>
<tr>
<th></th>
<th>Radix-2</th>
<th>Radix-4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constrained</td>
<td>Relaxed</td>
</tr>
<tr>
<td>Delay (ns)</td>
<td>2.18</td>
<td>14.17</td>
</tr>
<tr>
<td>Clock (MHz)</td>
<td>458</td>
<td>70</td>
</tr>
<tr>
<td>Throughput (Mbps)</td>
<td>460</td>
<td>70</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>0.279</td>
<td>0.165</td>
</tr>
</tbody>
</table>

Viterbi decoder achieves 55% and 100% higher throughput at a factor of 2 and 1.8 larger area for timing driven and relaxed synthesis respectively. Thus area penalty is affordable compared to the increase in throughput.

Table IV shows the detailed area breakdown for all sub-units of the Viterbi decoders for timing-driven and relaxed synthesis. It is clearly visible that in case of highly constrained synthesis, the worst area penalty occurs in the PMU. Due to the critical path in the PMU, timing constraint synthesis requires more hardware-intensive transformations. The BMU, trace-back logic, and the LIFO are much less critical in terms of delay and can meet the constraints without significant area penalty. During the timing-driven (constrained) synthesis, the radix-2 PMU becomes twice as large and the radix-4 more than 3 times larger in area compared to the relaxed case. In general, the radix-4 PMU leaves the synthesis tool somewhat more freedom for timing optimization since its logic is deeper and more complex than radix-2.

A factor that favors the radix-4 ACS is that it allows to achieve the same throughput at a lower clock frequency. This helps the other modules of the Viterbi decoder such as the survivor memory unit (SMU). The reason is that the trace-back based SMU design requires one or more SRAM(s) and the use of fast SRAM results in high area and power consumption. The design of the SMU unit is discussed in the next section.

Note that power dissipation aspects of the PMU have not been discussed so far. As it will be shown later, the power consumption of the PMU is a comparatively small fraction of the overall power budget for a Viterbi decoder. That is why, for the design of the ACS architecture minimizing power consumption should not play a dominating role. For low-power decoders, general optimization techniques to reduce power dissipation like clock gating can be applied to any of the PMU architectures introduced in this section [30], [31].

V. SURVIVOR SEQUENCE UPDATE AND STORAGE UNIT

A. Design Space

The survivor sequence update and storage unit (briefly called survivor memory unit, SMU) receives decisions from the PMU and produces the decoded sequence. Fig. 26 shows the design space for the SMU implementation. The implementation techniques for this unit can be divided in two classes: standard and adaptive.

TBD determines the number of memory access operations required by the SMU. The value of the TBD parameter depends on the transmission channel conditions (or the channel model

with tightest timing constraints settings possible to obtain a decoder with maximum throughput. Relaxed synthesis sets minimal timing constraints to get the lowest area figure. The results are summarized in Table III which also shows the overall area of the Viterbi decoder. Based on these figures, radix-4-based
used in the simulation). On noisy channels, trace-back paths need longer to merge, so (since usually a Viterbi decoder is designed for a worst case) TBD values become larger. On the other side, with fixed trace-back depth and good channel conditions (higher SNR), SMU may run redundant memory access operations, since paths tend to merge faster. Adaptive SMU implementations are used to reduce the number of these redundant accesses.

There are a number of adaptive techniques mentioned in the literature. Although they all require additional hardware to be added to the SMU, on average power dissipation of the unit can be reduced due to a lower number of memory accesses. In [32], [33], the adaptive techniques known as M-algorithm and T-algorithm are discussed, which reduces the number of states to consider by eliminating the states with too large path metric, basically dynamically adopting the number of ACS to calculate and trace-back paths to store. This idea is further developed in [39] as adaptive approximation by adding a pruning unit to the decoder’s core. The pruning unit limits the number of trace-back paths to follow, again aiming at power consumption reduction. Moreover, an SMU with dynamic prediction combined with trace-forward technique (see below) is introduced in [34] for Viterbi decoders used in wireless applications, which reduces the number of memory access by more than 70% in certain cases.

It is quite difficult to give general guidelines for the usage of adaptive techniques in the SMU, since their positive affects are very channel/application dependent (as mentioned in all references). For some designs, a very noisy channel can even lead to some data loss in the decoder, so additional hardware is required for recovery [32]. The only way to get figures for a specific design are experiments and simulation with accurate channel characteristics in every single application case. Unfortunately, all references known to the authors only include simulation data for AWGN channels, so the claimed power saving figures are questionable if, e.g., wireless channels with multipath and fading are considered. That is why the scope of this paper is limited to standard nonadaptive SMU designs. Nevertheless, for Viterbi decoder designs with strong focus on power reductions adaptive SMU is definitely something to consider, provided the designer is willing to spend some time on simulating the decoder with proper channel models. A detailed overview of adaptive trellis decoding techniques can be found in [40].

For standard (nonadaptive) Viterbi decoder there are two major techniques to implement the SMU: register exchange (RE) and trace-back (TB). RE uses a set of multiplexers and registers to store and update the survivor sequence of each state for trace-back depth (TBD) trellis stages. It is highly regular, area efficient and has low latency. But, due to frequent switching of states in the registers (bits are physically shifting from one stage to the next), it consumes a lot of power especially for larger values of TBD. The structure of the RE network repeats the trellis of the convolutional code as it is shown in Fig. 27 for a simple $K = 3$ code. Since it can be assumed that after TBD cycles, all survivor paths have merged, the output of any register from the last stage of the RE network can be used as the output of the Viterbi decoder.

The TB stores the decisions in an SRAM and reads them in reverse order during the trace-back cycle. As the update rate of the memory is much less frequent than the update rate of the registers in the RE technique, the power consumption is significantly reduced.

TB is split in three major operations. These are write, merge, and decode as shown in Fig. 28. If a survivor path is traced back for TBD stages, then the state at which all paths merge can be determined. Symbols traced back beyond this state can be used as decoded output. In each cycle 1 column of decisions is written into memory and $M$ columns are read for trace-back and decode. Pointers are used to keep track of the column read. The
1-pointer technique uses a single pointer multiple times to read $M$ columns. Multiple pointers are used in $M$-pointer technique for reading $M$-columns. The $M$-pointer technique requires inefficient and expensive multi-port or multi-bank memory, so it is rather unsuitable for a VLSI implementation.

Trace forward (TF) technique [4] is used to eliminate the merge stage (i.e., the buffer referred as the “merge block” in Fig. 28) 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. A refined version of this technique is introduced in [37].

Pre-traceback (PTB) [4] is a technique used to reduce memory access frequency. Instead of writing decisions at each stage processed by the PMU, $m$-stages are pre-traced and written as one composite decision. Thus, only one column of write operations is required for every $m$ 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 [40].

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. The area for both increases linearly with TBD and exponentially with $K$.

$$\text{SMU Area} \approx A \times \text{TBD} \times B \times 2^{(K-1)}$$

where $A$ is the number of trace-back memory blocks (4 with straightforward single-port-memory approach, 3 with folded read-write access, 2 if trace-forward is used, 1 for RE), $B$ is an area of a flip-flop plus multiplexer for RE or an area of an SRAM bit for TB respectively.

As can be seen, the area and critical path delay of the SMU are fixed by the technology. The most important decision for the designer is the architecture choice between RE and TB which is mainly driven by power consumption. Therefore the further analysis is completely dedicated to the power dissipation aspect.

**B. Implementation**

Simulations of power dissipation were ran for a radix-2 $K = 7$ Viterbi decoder with TBD of 128. The middle column of Table V shows the corresponding power dissipation profile. As the decision memory in the SMU is synthesized using a standard cell library, it has a high power consumption (110.9 mW compared to 6.1 mW for the rest of the SMU). Therefore for low power designs, the SMU should be optimized e.g., using low-power low-leakage SRAMs.

Such SRAM modules are available with power dissipation as low as 0.0324 mW/MHz. Using these memories in the Viterbi decoder from Table V would bring the power dissipation down to 22.3 mW at 250 MHz access frequency. In addition, it can be observed from Table VI that PTB occupy only 1.7% of the SMU area and dissipates only 1.9% of the total SMU power. This makes PTB an attractive solution to reduce the access frequency of the decision memory which enables the usage of low power high density SRAMs operating at a lower frequency.

The right column of Table V shows power dissipation profile for the Viterbi decoder using the RE technique for the SMU. In this case, 72% of the total power is dissipated in the SMU. Compared to the TB implementation which costs 57% even with non-optimal standard cell based memories, this is quite high. As discussed above, the size of the RE network is determined by $K$ and TBD. Therefore, we can conclude that for high values of $K$ and TBD, RE is not a suitable design choice in terms of power.

As already discussed, for decoders running at very high speed, SRAMs of matching clock frequency may not be
TABLE VIII
CRITERIA FOR OPTIMIZATION

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Memory Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_p &lt; \frac{f_m}{3} )</td>
<td>1 single-port memory+TF (optional)</td>
</tr>
<tr>
<td>( \frac{f_m}{3} &lt; f_p &lt; \frac{f_m}{2} )</td>
<td>1 single-port memory+TF</td>
</tr>
<tr>
<td>( \frac{f_m}{2} &lt; f_p &lt; f_m )</td>
<td>2 single-port memories+TF (( K &gt; 7 )) or 1 single-port memory+TF+2-level PTB (( K \leq 7 ))</td>
</tr>
<tr>
<td>( f_m &lt; f_p &lt; 2 \times f_m )</td>
<td>2 ( N )-bit single-port memories+TF+2-4-level PTB (( K &gt; 7 )) or 2 ( N )-bit 1 single-port memory+TF+4-level PTB (( K \leq 7 ))</td>
</tr>
</tbody>
</table>

TABLE IX
SUMMARY OF THE CHARTS, FIGURES, AND TABLES

<table>
<thead>
<tr>
<th>Design Aspect</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design space</td>
<td>Fig. 6 (top view), 7 (PMU), and 26 (SMU)</td>
</tr>
<tr>
<td>Algorithmic and word-level optimization techniques</td>
<td>Table I</td>
</tr>
<tr>
<td>PMU area and delay for different adder architectures</td>
<td>Table II</td>
</tr>
<tr>
<td>PMU area and delay as a function of softbits</td>
<td>Fig. 15, 16, and 17</td>
</tr>
<tr>
<td>Comparison between radix-2 and radix-4 PMUs</td>
<td>Fig. 18, 19, 20, and 21</td>
</tr>
<tr>
<td>softbits and ( K ) vs. critical path and area for radix-2 PMUs</td>
<td>Fig. 22 and 24 (3D-charts)</td>
</tr>
<tr>
<td>softbits and ( K ) vs. critical path and area for radix-4 PMUs</td>
<td>Fig. 23 and 25 (3D-charts)</td>
</tr>
<tr>
<td>Radix-2 vs. radix-4 comparison for a complete decoder</td>
<td>Table III and IV</td>
</tr>
<tr>
<td>SMU power and area figures</td>
<td>Table V and VI</td>
</tr>
<tr>
<td>Guidelines for SMU memory architecture choice</td>
<td>Table VIII</td>
</tr>
</tbody>
</table>

available. The memory architecture depends on the frequency at which the PMU is running and the maximum access frequency of the SRAMs. Table VIII summarizes the guidelines for the memory architecture choice depending on the clock frequencies. \( f_m \) denotes the maximum access frequency for the SRAMs. Operation frequency of the PMU is denoted by \( f_p \), which implies the frequency at which the decision columns are generated.

VI. CONCLUSION

In this paper, a comprehensive analysis of the Viterbi decoder design space is presented. Table VII summarizes the importance of the different subunits of the decoder depending on the optimization criteria (the BMU is not mentioned, since its influence on area, power and throughput is negligible). As shown in Table VII, the PMU is critical for throughput while the SMU is critical for latency and power consumption.

The most significant contributions of the paper can be summarized as follows:

- detailed presentation of the design space for the hard-decision Viterbi decoder and each of its subunits;
- comparison of different adder architectures for the PMU implementation (ripple-carry, carry-lookahead, and bit-serial);
- quantitative comparison of behavioral and structural HDL coding styles;
- quantitative comparison of different ACS architectures; in particular, it has been shown that the general assumption of CSA being better than ACS and the radix-4 technique being inefficient in terms of area does not hold for some cases;
- analysis of the influence of timing constraints (relaxed vs. constrained synthesis) on the design choices;
- detailed analysis of the impact of constraint length and input bitwidth on area and throughput;
- detailed analysis of different SMU implementation techniques and their impact on power consumption based on post-layout simulation.

For a better overview of the material, Table IX summarizes the figures, charts, and tables related to different design aspects to help the reader in finding the proper guidelines for choosing the Viterbi decoder design.

So far, this work discusses most of the known VLSI implementation techniques for the hard-decision Viterbi algorithm in standard cell CMOS technology and carefully analyzes the tradeoffs and dependencies between different design decisions. To the best of authors’ knowledge, this is the most comprehensive analysis of hard-decision Viterbi algorithm VLSI implementation based on actual designs including post-layout experiments published so far.

ACKNOWLEDGMENT

The authors would like to thank Dr. N. Engin for valuable discussions on hardware-software tradeoffs in Viterbi implementation and A. Vaassen for supporting the layout experiments.

REFERENCES
