High-Speed Low-Power Viterbi Decoder Design for TCM Decoders

Jinjin He, Huaping Liu, Zhongfeng Wang, Xinming Huang, and Kai Zhang

***Abstract*—High-speed, low-power design of Viterbi decoders for trellis coded modulation (TCM) systems is presented in this paper. It is well known that the Viterbi decoder (VD) is the dominant module determining the overall power consumption of TCM decoders. We propose a pre-computation architecture incorporated with T-algorithm for VD, which can effectively reduce the power consumption without degrading the decoding speed much. A general solution to derive the optimal pre-computation steps is also given in the paper. Implementation result of a VD for a rate-3/4 convolutional code used in a TCM system shows that compared with the full trellis VD, the precomputation architecture reduces the power consumption by as much as 70% without performance loss, while the degradation in clock speed is negligible.**

***Index Terms*—Trellis coded modulation (TCM), viterbi decoder, VLSI.**

I. Introduction

Trellis coded modulation (TCM) schemes are used in many bandwidth-efficient systems. Typically, a TCM system employs a high-rate convolutional code, which leads to a high complexity of the Viterbi decoder (VD) for the TCM decoder, even if the constraint length of the convolutional code is moderate. For example, the rate-3/4 convolutional code used in a 4-D TCM system for deep space communications [1] has a constraint length of 7; however, the computational complexity of the corresponding VD is equivalent to that of a VD for a rate-1/2 convolutional code with a constraint length of 9 due to the large number of transitions in the trellis. Therefore, in terms of power consumption, the Viterbi decoder is the dominant module in a TCM decoder. In order to reduce the computational complexity as well as the power consumption, low-power schemes should be exploited for the VD in a TCM decoder.

General solutions for low-power VD design have been well studied by existing work. Power reduction in VDs could be achieved by reducing the number of states (for example, reduced-state sequence decoding (RSSD) [2], M-algorithm [3] and T-algorithm [4], [5]) or by over-scaling the supply voltage [6]. Over-scaling of the supply voltage usually needs to take into consideration the whole system that includes the VD (whether the system allows such an over-scaling or not), which is not the main focus of our research. RSSD is in general not as efficient as the M-algorithm [2] and T-algorithm is more commonly used than M-algorithm in practical applications, because the M-algorithm requires a sorting process in a feedback loop while T-algorithm only searches for the optimal path metric (PM), that is, the minimum value or the maximum value of all PMs.

Manuscript received July 28, 2010; revised January 15,2011; accepted January 19, 2011. Date of publication March 07, 2011; date of current version March 12, 2012. This paper was presented in part at the 43rd Asilomar Conference on Signal, Systems and Computers, Pacific Grove, CA, 2009.

J. He and H. Liu are with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331 USA (e-mail: hejin@eecs.oregonstate.edu; hliu@eecs.oregonstate.edu).

Z. Wang is with the Broadcom Corporation, Irvine, CA 92617 USA (e-mail: zfwang@broadcom.com).

X. Huang and K. Zhang are with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609 USA (e-mail: xhuang@wpi.edu; kzhang@wpi.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2111392

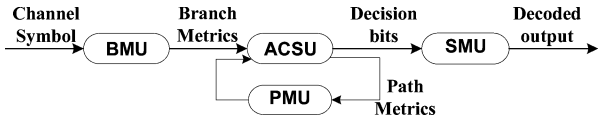

Fig. 1. Functional diagram of a viterbi decoder.

T-algorithm has been shown to be very efficient in reducing the power consumption [7], [8]. However, searching for the optimal PM in the feedback loop still reduces the decoding speed. To overcome this drawback, two variations of the T-algorithm have been proposed: the relaxed adaptive VD [7], which suggests using an estimated optimal PM, instead of finding the real one each cycle and the limited-search parallel state VD based on scarce state transition (SST) [8]. In our preliminary work [9], we have shown that when applied to high-rate convolutional codes, the relaxed adaptive VD suffers a severe degradation of bit-error-rate (BER) performance due to the inherent drifting error between the estimated optimal PM and the accurate one. On the other hand, the SST based scheme requires predecoding and re-encoding processes and is not suitable for TCM decoders. In TCM, the encoded data are always associated with a complex multi-level modulation scheme like 8-ary phase-shift keying (8PSK) or 16/64-ary quadrature amplitude modulation (16/64QAM) through a constellation point mapper. At the receiver, a soft-input VD should be employed to guarantee a good coding gain. Therefore, the computational overhead and decoding latency due to predecoding and re-encoding of the TCM signal become high. In our preliminary work [9], we proposed an add-compare-select unit (ACSU) architecture based on precomputation for VDs incorporating T-algorithm, which efficiently improves the clock speed of a VD with T-algorithm for a rate-3/4 code. In this work, we further analyze the precomputation algorithm. A systematic way to determine the optimal precomputation steps is presented, where the minimum number of steps for the critical path to achieve the theoretical iteration bound is calculated and the computational complexity overhead due to pre-computation is evaluated. Then, we discuss a complete low-power high-speed VD design for the rate-3/4 convolutional code [1]. Finally ASIC implementation results of the VD are reported, which have not been obtained in our previous work in [9].

The remainder of this paper is organized as follows. Section II gives the background information of VDs. Section III presents the precomputation architecture with T-algorithm and discusses the choice of precomputation steps. Details of a design example including the modification of survivor-path memory unit (SMU) are discussed in Section III. Synthesis and power estimation results are reported in Section IV and conclusions are given in Section V.

II. Viterbi Decoder

A typical functional block diagram of a Viterbi decoder is shown in Fig. 1. First, branch metrics (BMs) are calculated in the BM unit (BMU) from the received symbols. In a TCM decoder, this module is replaced by transition metrics unit (TMU), which is more complex than the BMU. Then, BMs are fed into the ACSU that recursively computes the PMs and outputs decision bits for each possible state transition. After that, the decision bits are stored in and retrieved from the SMU in order to decode the source bits along the final survivor path. The PMs of the current iteration are stored in the PM unit (PMU).

T-algorithm requires extra computation in the ACSU loop for calculating the optimal PM and puncturing states. Therefore, a straightforward implementation of T-algorithm will dramatically reduce the

decoding speed. The key point of improving the clock speed of T-algorithm is to quickly find the optimal PM.

III. Precomputation Architecture

A. Precomputation Algorithm

The basic idea of the precomputation algorithm was presented in [9]. Consider a VD for a convolutional code with a constraint length k, where each state receives p candidate paths. First, we expand PMs at the current time slot n ($\mathrm{PMs}(n)$) as a function of $\mathrm{PMs}(n-1)$ to form a look-ahead computation of the optimal $\mathrm{PM}-\mathrm{PM}_{\mathrm{opt}}(n)$. If the branch metrics are calculated based on the Euclidean distance, $\mathrm{PM}_{\mathrm{opt}}(n)$ is the minimum value of $\mathrm{PMs}(n)$ obtained as

$$\begin{aligned}
&\mathrm{PM}_{\mathrm{opt}}(n)\\
&= \min\{\mathrm{PM}_0(n), \mathrm{PM}_1(n), \ldots\ldots\ldots, \mathrm{PM}_{2-1}^{k}(n)\}\\
&= \min\{\min[\mathrm{PM}_{0,0}(n-1) + \mathrm{BM}_{0,0}(n),\\
&\qquad \mathrm{PM}_{0,1}(n-1) + \mathrm{BM}_{0,1}(n), \ldots\ldots,\\
&\qquad \mathrm{PM}_{0,p}(n-1) + \mathrm{BM}_{0,p}(n)],\\
&\quad \min[\mathrm{PM}_{1,0}(n-1) + \mathrm{BM}_{1,0}(n),\\
&\qquad \mathrm{PM}_{1,1}(n-1) + \mathrm{BM}_{1,1}(n), \ldots\ldots,\\
&\qquad \mathrm{PM}_{1,p}(n-1) + \mathrm{BM}_{1,p}(n)],\\
&\qquad \ldots\ldots\ldots\ldots\ldots\ldots,\\
&\quad \min[\mathrm{PM}_{2^{k-1}-1,0}(n-1) + \mathrm{BM}_{2^{k-1}-1,0}(n),\\
&\qquad \mathrm{PM}_{2^{k-1}-1,1}(n-1) + \mathrm{BM}_{2^{k-1}-1,1}(n), \ldots\ldots,\\
&\qquad \mathrm{PM}_{2^{k-1}-1,p}(n-1) + \mathrm{BM}_{2^{k-1}-1,p}(n)]\}\\
&= \min\{\mathrm{PM}_{0,0}(n-1) + \mathrm{BM}_{0,0}(n),\\
&\quad \mathrm{PM}_{0,1}(n-1) + \mathrm{BM}_{0,1}(n), \ldots\ldots,\\
&\quad \mathrm{PM}_{0,p}(n-1) + \mathrm{BM}_{0,p}(n),\\
&\quad \mathrm{PM}_{1,0}(n-1) + \mathrm{BM}_{1,0}(n),\\
&\quad \mathrm{PM}_{1,1}(n-1) + \mathrm{BM}_{1,1}(n), \ldots\ldots,\\
&\quad \mathrm{PM}_{1,p}(n-1) + \mathrm{BM}_{1,p}(n),\\
&\quad \ldots\ldots\ldots\ldots\ldots\ldots,\\
&\quad \mathrm{PM}_{2^{k-1}-1,0}(n-1) + \mathrm{BM}_{2^{k-1}-1,0}(n),\\
&\quad \mathrm{PM}_{2^{k-1}-1,1}(n-1) + \mathrm{BM}_{2^{k-1}-1,1}(n), \ldots\ldots,\\
&\quad \mathrm{PM}_{2^{k-1}-1,p}(n-1) + \mathrm{BM}_{2^{k-1}-1,p}(n)\}. \qquad (1)
\end{aligned}$$

Then, we group the states into several clusters to reduce the computational overhead caused by look-ahead computation. The trellis butterflies for a VD usually have a symmetric structure. In other words, the states can be grouped into m clusters, where all the clusters have the same number of states and all the states in the same cluster will be extended by the same BMs. Thus, (1) can be rewritten as

$$\begin{aligned}
\mathrm{PM}_{\mathrm{opt}}(n) = \min\{&\min(\mathrm{PMs}\,(n-1)\text{ in cluster }1)\\
&+ \min(\mathrm{BMs}(n)\text{ for cluster }1),\\
&\min(\mathrm{PMs}\,(n-1)\text{ in cluster }2)\\
&+ \min(\mathrm{BMs}(n)\text{ for cluster }2),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots,\\
&\min(\mathrm{PMs}\,(n-1)\text{ in cluster }m)\\
&+ \min(\mathrm{BMs}(n)\text{ for cluster }m)\}. \qquad (2)
\end{aligned}$$

The $\min(\mathrm{BMs})$ for each cluster can be easily obtained from the BMU or TMU and the $\min(\mathrm{PMs})$ at time $n-1$ in each cluster can be precalculated at the same time when the ACSU is updating the new PMs for time n. Theoretically, when we continuously decompose $\mathrm{PMs}(n-1)$, $\mathrm{PMs}(n-2)$, ..., the precomputation scheme can be extended to q steps, where q is any positive integer that is less than n. Hence, $\mathrm{PM}_{\mathrm{opt}}(n)$ can be calculated directly from $\mathrm{PMs}(n-q)$ in q cycles.

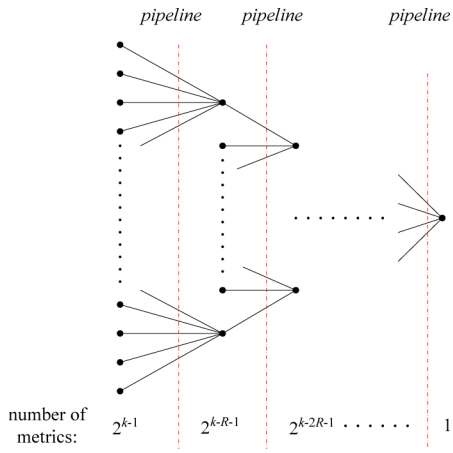

Fig. 2. Topology of precomputation pipelining.

B. Choosing Precomputation Steps

In [9], we have shown through a design example that, q-step precomputation can be pipelined into q stages, where the logic delay of each stage is continuously reduced as q increases. As a result, the decoding speed of the low-power VD is greatly improved. However, after reaching a certain number of steps, q_b, further precomputation would not result in additional benefits because of the inherent iteration bound of the ACSU loop. Therefore, it is worth to discuss the optimal number of precomputation steps.

In a TCM system, the convolutional code usually has a coding rate of $R/(R+1)$, $R = 2, 3, 4, \ldots$, so that in (1), $p = 2^R$ and the logic delay of the ACSU is $T_{\mathrm{ACSU}} = T_{\mathrm{adder}} + T_{\mathrm{p-in\_comp}}$, where T_{adder} is the logic delay of the adder to compute PMs of each candidate path that reaches the same state and $T_{\mathrm{p-in\_comp}}$ is the logic delay of a p-input comparator to determine the survivor path (the path with the minimum metric) for each state. If T-algorithm is employed in the VD, the iteration bound is slightly longer than T_{ACSU} because there will be another two-input comparator in the loop to compare the new PMs with a threshold value obtained from the optimal PM and a preset T as shown in (3)

$$T_{\mathrm{bound}} = T_{\mathrm{adder}} + T_{p-in\_comp} + T_{2-in\_comp}. \qquad (3)$$

To achieve the iteration bound expressed in (3), for the precomputation in each pipelining stage, we limit the comparison to be among only p or $2p$ metrics. To simplify our evaluation, we assume that each stage reduces the number of the metrics to $1/p$ (or 2^{-R}) of its input metrics as shown in Fig. 2. The smallest number of precomputation steps (q_b) meeting the theoretical iteration bound should satisfy $(2^R)^{q_b} \geq 2^{k-1}$. Therefore, $q_b \geq (k-1)/R$ and q_b is expressed as (4), where $\lceil\cdot\rceil$ denotes the ceiling function

$$q_b = \left\lceil \frac{k-1}{R} \right\rceil. \qquad (4)$$

In the design example shown in [9], with a coding rate of 3/4 and constraint length of 7, the minimum precomputation steps for the VD to meet the iteration bound is 2 according to (4). It is the same value as we obtained from direct architecture design [9]. In some cases, the number of remaining metrics may slightly expand during a certain pipeline stage after addition with BMs. Usually, the extra delay can be absorbed by an optimized architecture or circuit design. Even if the extra delay is

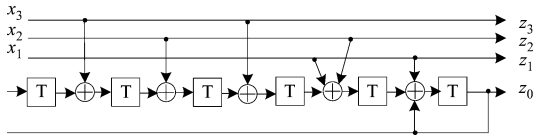

Fig. 3. Rate 3/4 convolutional encoder.

hard to eliminate, the resultant clock speed is very close to the theoretical bound. To fully achieve the iteration bound, we could add another pipeline stage, though it is very costly as will be shown next.

Computational overhead (compared with conventional T-algorithm) is an important factor that should be carefully evaluated. Most of the computational overhead comes from adding BMs to the metrics at each stage as indicated in (2). In other words, if there are m remaining metrics after comparison in a stage, the computational overhead from this stage is at least m addition operations. The exact overhead varies from case to case based on the convolutional code's trellis diagram. Again, to simplify the evaluation, we consider a code with a constraint length k and q precomputation steps. Also, we assume that each remaining metric would cause a computational overhead of one addition operation. In this case, the number of metrics will reduce at a ratio of $2^{(k-1)/q}$ and the overall computational overhead is (measured with addition operation)

$$N_{overhead} = 2^0 + 2^{(k-1)/q} + 2^{2(k-1)/q} \cdots + 2^{(q-1)(k-1)/q} = \frac{2^{q\cdot(k-1)/q} - 1}{2^{(k-1)/q} - 1} = \frac{2^{k-1} - 1}{2^{(k-1)/q} - 1}. \quad (5)$$

The estimated computational overhead according to (5) is $63/(2^{6/q} - 1)$ when $k = 7$ and $q \leq 6$, which increases almost exponentially to q. In a real design, the overhead increases even faster than what is given by (5) when other factors (such as comparisons or expansion of metrics as we mentioned above) are taken into consideration. Therefore, a small number of precomputational steps is preferred even though the iteration bound may not be fully satisfied. In most cases, one- or two-step precomputation is a good choice.

The above analysis also reveals that precomputation is not a good option for low-rate convolutional codes (rate of $1/Rc$, $Rc = 2, 3, \ldots$), because it usually needs more than two steps to effectively reduce the critical path (in that case, $R = 1$ in (4) and q_b is $k - 1$). However, for TCM systems, where high-rate convolutional codes are always employed, two steps of precomputation could achieve the iteration bound or make a big difference in terms of clock speed. In addition, the computational overhead is small.

IV. Low-Power High-Speed Viterbi Decoder Design

We still use the 4-D 8PSK TCM system described in [1] as the example. The rate-3/4 convolutional code employed in the TCM system is shown in Fig. 3. Preliminary BER performance and architecture design for the ACSU unit have been discussed in [9]. In this section, we further address the SMU design issue. Later in the next section, we will report ASIC implementation results that have not been obtained before.

BER performance of the VD employing T-algorithm with different values of T over an additive white Gaussian noise channel is shown in Fig. 4. The simulation is based on a 4-D 8PSK TCM system employing the rate-3/4 code [10]. The overall coding rate is 11/12 after due to other uncoded bits in TCM system. Compared with the ideal Viterbi algorithm, the threshold "$T\mathrm{pm}$" can be lowered to 0.3 with less than 0.1 dB of performance loss, while the computational complexity could be reduced by up to 90% [9] ideally. Since the precomputation algorithm always finds the accurate optimal PM, its BER performance is the same as that of the conventional T-algorithm.

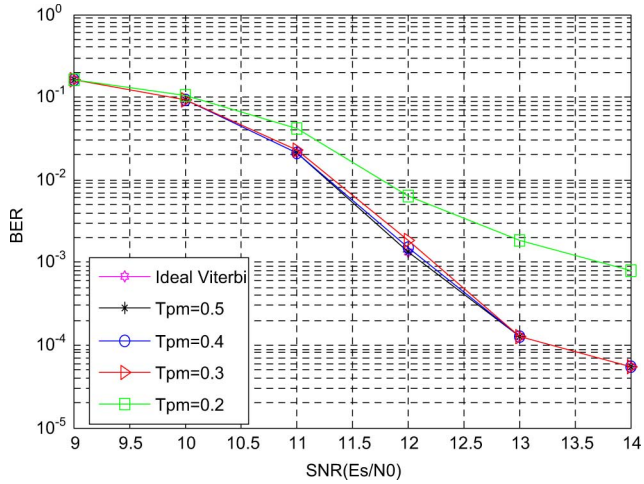

Fig. 4. BER performance of T-algorithm.

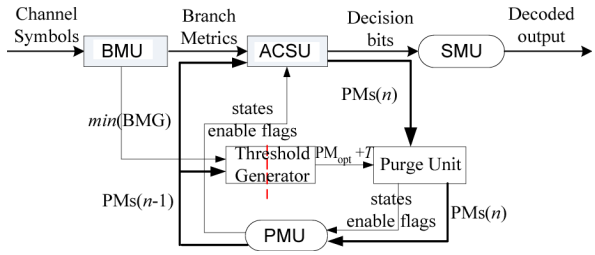

Fig. 5. VD with two-step precomputation T-algorithm.

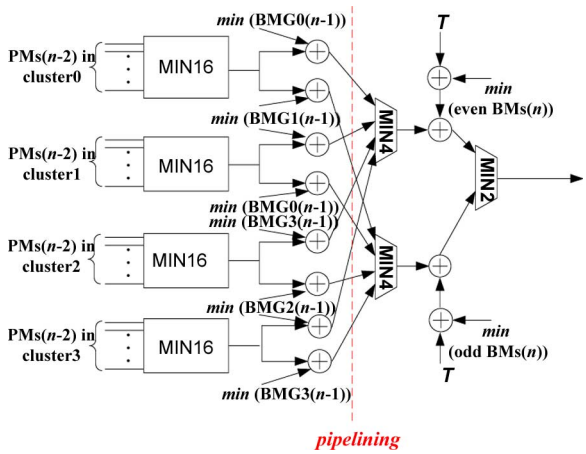

Fig. 6. Architecture of TGU.

A. ACSU Design

We have concluded that two-step precomputation is the optimal choice for the rate-3/4 code VD. For convenience of discussion, we define the left-most register in Fig. 3 as the most-significant-bit (MSB) and the right-most register as the least-significant-bit (LSB). The 64 states and PMs are labeled from 0 to 63. The two-step precomputation is expressed as (6) [9]

$$\begin{aligned}\mathrm{PM}_{\mathrm{opt}}(n) = \\ \min[\min\{&\min(\mathrm{cluster0}\ (n-2)) + \min(\mathrm{BMG0}(n-1)),\\ &\min(\mathrm{cluster1}\ (n-2)) + \min(\mathrm{BMG1}(n-1)),\\ &\min(\mathrm{cluster2}\ (n-2)) + \min(\mathrm{BMG3}(n-1)),\\ &\min(\mathrm{cluster3}\ (n-2)) + \min(\mathrm{BMG2}(n-1))\}\\ &+ \min(\mathrm{even\ BMs}(n)),\end{aligned}$$

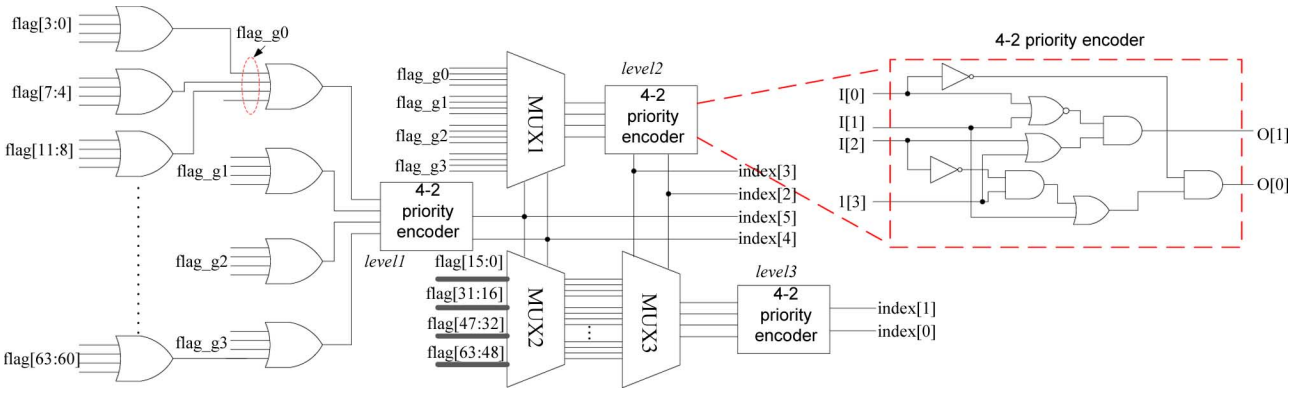

Fig. 7. Architecture of 64-to-6 priority encoder.

$$\begin{aligned}\min\{&\min(\text{cluster0}\ (n-2)) + \min(\text{BMG1}(n-1)),\\ &\min(\text{cluster1}\ (n-2)) + \min(\text{BMG0}(n-1)),\\ &\min(\text{cluster2}\ (n-2)) + \min(\text{BMG2}(n-1)),\\ &\min(\text{cluster3}\ (n-2)) + \min(\text{BMG3}(n-1))\}\\ &+ \min(\text{odd BMs}(n))] \end{aligned} \tag{6}$$

where

$$\begin{aligned}\text{cluster0} &= \{\text{PM}_m | 0 \le m \le 63,\ m \bmod 4 = 0\};\\ \text{cluster1} &= \{\text{PM}_m | 0 \le m \le 63,\ m \bmod 4 = 2\};\\ \text{cluster2} &= \{\text{PM}_m | 0 \le m \le 63,\ m \bmod 4 = 1\};\\ \text{cluster3} &= \{\text{PM}_m | 0 \le m \le 63,\ m \bmod 4 = 3\};\\ \text{BMG0} &= \{\text{BM}_m | 0 \le m \le 15,\ m \bmod 4 = 0\};\\ \text{BMG1} &= \{\text{BM}_m | 0 \le m \le 15,\ m \bmod 4 = 2\};\\ \text{BMG2} &= \{\text{BM}_m | 0 = m = 15,\ m \bmod 4 = 1\};\\ \text{BMG3} &= \{\text{BM}_m | 0 \le m \le 15, m \bmod 4 = 3\}.\end{aligned}$$

The functional block diagram of the VD with two-step precomputation T-algorithm is shown in Fig. 5. The minimum value of each BM group (BMG) can be calculated in BMU or TMU and then passed to the "Threshold Generator" unit (TGU) to calculate $(\text{PM}_{\text{opt}} + T) \cdot (\text{PM}_{\text{opt}} + T)$ and the new PMs are then compared in the "Purge Unit" (PU). The architecture of the TGU is shown in Fig. 6, which implements the key functions of the two-step precomputation scheme.

In Fig. 6, the "MIN 16" unit for finding the minimum value in each cluster is constructed with two stages of four-input comparators. This architecture has been optimized to meet the iteration bound [9]. Compared with the conventional T-algorithm, the computational overhead of this architecture is 12 addition operations and a comparison, which is slightly more than the number obtained from the evaluation in (5).

B. SMU Design

In this section, we address an important issue regarding SMU design when T-algorithm is employed. There are two different types of SMU in the literature: register exchange (RE) and trace back (TB) schemes. In the regular VD without any low-power schemes, SMU always outputs the decoded data from a fixed state (arbitrarily selected in advance) if RE scheme is used, or traces back the survivor path from the fixed state if TB scheme is used, for low-complexity purpose. For VD incorporated with T-algorithm, no state is guaranteed to be active at all clock cycles. Thus it is impossible to appoint a fixed state for either outputting the decoded bit (RE scheme) or starting the trace-back process (TB scheme). In the conventional implementation of T-algorithm, the decoder can use the optimal state (state with PM_{opt}), which is always enabled, to output or trace back data. The process of searching for PM_{opt} can find out the index of the optimal state as a byproduct. However, when the estimated PM_{opt} is used [8], or in our case PM_{opt} is calculated from PMs at the previous time slot, it is difficult to find the index of the optimal state.

TABLE I
TRUTH TABLE OF 64-TO-6 PRIORITY ENCODER

flag[63:0]	index[5:0]
x x .....x x x x x 1	0 0 0 0 0 0
x x .....x x x x 1 0	0 0 0 0 0 1
x x .....x x x 1 0 0	0 0 0 0 1 0
x x .....x x 1 0 0 0	0 0 0 0 1 1
x x .....x 1 0 0 0 0	0 0 0 1 0 0
⋮	⋮
x 1 0 .....0 0 0 0 0 0	1 1 1 1 1 0
1 0 0 .....0 0 0 0 0 0	1 1 1 1 1 1

A practical method is to find the index of an enabled state through a $2^{(k-1)}$-to-$(k-1)$ priority encoder. Suppose that we have labeled the states from 0 to 63. The output of the priority encoder would be the unpurged state with the lowest index. Assuming the purged states have the flag "0" and other states are assigned the flag "1", the truth table of such a priority encoder is shown in Table I, where "flag" is the input and "index" is the output.

Implementation of such a table is not trivial. In our design, we employ an efficient architecture for the 64-to-6 priority encoder based on three 4-to-2 priority encoders, as shown in Fig. 7. The 64 flags are first divided into 4 groups, each of which contains 16 flags. The priority encoder at level 1 detects which group contains at least one "1" and determines "$\text{index}[5:4]$". Then MUX2 selects one group of flags based on "$\text{index}[5:4]$". The input of the priority encoder at level 2 can be computed from the output of MUX2 by "OR" operations. We can also reuse the intermediate results by introducing another MUX (MUX1). The output of the priority encoder at level 2 is "$\text{index}[3:2]$". Again, "$\text{index}[3:2]$" selects four flags (MUX3) as the input of the priority encoder at level 3. Finally, the last encoder will determine "$\text{index}[1:0]$".

Implementing the 4-to-2 priority encoder is much simpler than implementing the 64-to-6 priority encoder. Its truth table is shown in Table II and the corresponding logics are shown in (7) and (8)

$$\begin{aligned}O[0] &= \overline{I[0]} \cdot (I[1] + I[3] \cdot \overline{I[2]} \cdot \overline{I[1]})\\ &= \overline{I[0]} \cdot (I[1] + I[3] \cdot \overline{I[2]});\end{aligned} \tag{7}$$

$$\begin{aligned}O[1] &= \overline{I[0]} \cdot \overline{I[1]} \cdot (I[2] + \overline{I[2]} I[3])\\ &= \overline{I[0] + I[1]} \cdot (I[2] + I[3]).\end{aligned} \tag{8}$$

TABLE II
TRUTH TABLE OF 4-TO-2 PRIORITY ENCODER

input (I [3:0])	output (O [1:0])
x x x 1	0 0
x x 1 0	0 1
x 1 0 0	1 0
1 0 0 0	1 1

TABLE III
SYNTHESIS RESULTS FOR MAXIMUM CLOCK SPEED

	Max speed (MHz)	cell area (mm^2)
Full-trellis VD	505	0.58
VD with 2-step pre-computation	446.4 (-11.6%)	0.68 (+17.2%)
Conventional T-algorithm	232 (-54.1%)	0.685 (+18%)

TABLE IV
POWER ESTIMATION RESULTS

	Power (mw)	
Full-trellis VD	21.473 (100%)	
VD with 2-step pre-computation architecture	Tpm = 0.75	20.069 (93.5%)
	Tpm = 0.625	17.186 (80.0%)
	Tpm = 0.5	11.754 (54.7%)
	Tpm = 0.375	6.6127 (30.8%)

V. IMPLEMENTATION RESULTS

The full-trellis VD, the VD with the two-step precomputation architecture and one with the conventional T-algorithm are modeled with Verilog HDL code. The soft inputs of all VDs are quantized with 7 bits. Each PM in all VDs is quantized as 12 bits. RE scheme with survival length of 42 is used for SMU and the register arrays associated with the purged states are clock-gated to reduce the power consumption in SMU. For ASIC synthesis, we use TSMC 90-nm CMOS standard cell. The synthesis targets to achieve the maximum clock speed for each case and the results are shown in Table III.

Table III shows that the VD with two-step precomputation architecture only decreases the clock speed by 11% compared with the full-trellis VD. Meanwhile, the increase of the hardware area is about 17%. The decrease of clock speed is inevitable since the iteration bound for VD with T-algorithm is inherently longer than that of the full-trellis VD. Also, any kinds of low-power scheme would introduce extra hardware like the purge unit shown in Fig. 5 or the clock-gating module in the SMU. Therefore, the hardware overhead of the proposed VD is expected. On the other hand, the VD with conventional T-algorithm cannot achieve half of the clock speed of the full trellis VD. Therefore, for high-speed applications, it should not be considered. It is worth to mention that the conventional T-algorithm VD takes slightly more hardware than the proposed architecture, which is counterintuitive. This is because the former decoder has a much longer critical path and the synthesis tool took extra measures to improve the clock speed. It is clear that the conventional T-algorithm is not suitable for high-speed applications. If the target throughput is moderately high, the proposed architecture can operate at a lower supply voltage, which will lead to quadratic power reduction compared to the conventional scheme. Thus we next focus on the power comparison between the full trellis VD and the proposed scheme.

We estimate the power consumption of these two designs with Synopsys Prime Power under the clock speed of 200 Mb/s (power supply of 1.0 V, temperature of 300 K). A total of 1133 received symbols (12 000 bits) are simulated. The results are shown in Table IV.

With the finite word-length implementation, the threshold can only be changed by a step of 0.125. Therefore, to maintain a good BER performance, the minimum threshold we chose is 0.375. Table IV shows that, as the threshold decreases, the power consumption of the proposed VD is reduced accordingly. In order to achieve the same BER performance, the proposed VD only consumes 30.8% the power of the full-trellis VD.

VI. CONCLUSION

We have proposed a high-speed low-power VD design for TCM systems. The precomputation architecture that incorporates T-algorithm efficiently reduces the power consumption of VDs without reducing the decoding speed appreciably. We have also analyzed the precomputation algorithm, where the optimal precomputation steps are calculated and discussed. This algorithm is suitable for TCM systems which always employ high-rate convolutional codes. Finally, we presented a design case. Both the ACSU and SMU are modified to correctly decode the signal. ASIC synthesis and power estimation results show that, compared with the full-trellis VD without a low-power scheme, the precomputation VD could reduce the power consumption by 70% with only 11% reduction of the maximum decoding speed.

REFERENCES

[1] "Bandwidth-efficient modulations," Consultative Committee For Space Data System, Matera, Italy, CCSDS 401(3.3.6) Green Book, Issue 1, Apr. 2003.

[2] J. B. Anderson and E. Offer, "Reduced-state sequence detection with convolutional codes," *IEEE Trans. Inf. Theory*, vol. 40, no. 3, pp. 965–972, May 1994.

[3] C. F. Lin and J. B. Anderson, "M-algorithm decoding of channel convolutional codes," presented at the Princeton Conf. Info. Sci. Syst., Princeton, NJ, Mar. 1986.

[4] S. J. Simmons, "Breadth-first trellis decoding with adaptive effort," *IEEE Trans. Commun.*, vol. 38, no. 1, pp. 3–12, Jan. 1990.

[5] F. Chan and D. Haccoun, "Adaptive viterbi decoding of convolutional codes over memoryless channels," *IEEE Trans. Commun.*, vol. 45, no. 11, pp. 1389–1400, Nov. 1997.

[6] R. A. Abdallah and N. R. Shanbhag, "Error-resilient low-power viterbi decoder architectures," *IEEE Trans. Signal Process.*, vol. 57, no. 12, pp. 4906–4917, Dec. 2009.

[7] J. Jin and C.-Y. Tsui, "Low-power limited-search parallel state viterbi decoder implementation based on scarece state transition," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 11, pp. 1172–1176, Oct. 2007.

[8] F. Sun and T. Zhang, "Low power state-parallel relaxed adaptive viterbi decoder design and implementation," in *Proc. IEEE ISCAS*, May 2006, pp. 4811–4814.

[9] J. He, H. Liu, and Z. Wang, "A fast ACSU architecture for viterbi decoder using T-algorithm," in *Proc. 43rd IEEE Asilomar Conf. Signals, Syst. Comput.*, Nov. 2009, pp. 231–235.

[10] J. He, Z. Wang, and H. Liu, "An efficient 4-D 8PSK TCM decoder architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 808–817, May 2010.