# Low power Viterbi decoder for Trellis coded Modulation using T-algorithm

Md.Javeed, B.Sri lakshmi

**Abstract:** The viterbi decoder which is low power with the convolutional encoder for the trellis coded modulation is shown in this paper. Convolutional encoding with Viterbi decoding is a good forward error correction technique suitable for channels affected by noise degradation. In this paper it shows the viterbi decoder architecture with convolutional encoder with proposed precomputation T-algorithm which can effectively reduce the power consumption with negligible decrease in the speed. Implementation result is for ¾ convolutional code rate with constraint length 7 used for trellis coded modulation. This architecture reduces the power consumption up to 70% without any performance loss, while the degradation in clock speed is negligible.

**Key words**: Convolutional code, T-algorithm, Trellis coded modulation (TCM), viterbi decoder, VLSI.

## I. INTRODUCTION:

The use of convolutional codes with probabilistic decoding can significantly improve the error performance of a communication system [1]. Trellis coded modulation schemes are used in many bandwidth efficient systems. Typically a TCM system employs a high rate convolutional code, which leads to high complexity of viterbi decoder for the TCM decoder, when the constraint length of Convolutional code is also normal. For example the rate ¾ convolutional code used in trellis coded modulation system for any application has a constraint length of 7 will be in the complexity of the corresponding viterbi decoder for a rate ½ convolutional code with constraint length of 9 [2] due to the large number of transitions in the trellis. So, In terms of power consumption, the viterbi decoder is dominant module in a TCM decoder. In order to reduce the computational complexity as well as power consumption, low power schemes should be exploited for the VD in a TCM decoder.

General solutions for low power viterbi decoder design will be studied in our implementation work. Power reduction in VDs could be achieved by reducing the number of states, (for example reduced state sequence decoding [3], M-algorithm [4] and T-algorithm [1],[5],) or by over scaling the supply voltage[6].Over scaling of the supply voltage is having a problem that it needs to take whole system into consideration including with VD at which we are not focusing of our research. In practical application RSSD is more commonly used than M-Algorithm which is generally not as efficient as M-algorithm[3] and T-Algorithm. Basically M-Algorithm requires a sorting process in a feedback loop where as T–

Algorithm only searches for the optimal path metric [P] that is the maximum value or the minimum value of Ps.

T-Algorithm has been shown to very efficient in reducing the power consumption [7],[8]. However, searching for the optimal path metric in the feedback loop still reduces the decoding speed. To overcome this drawback, T-Algorithm has proposed in two variations, the relaxed adaptive VD [7], Which suggests using an estimated optimal path metric, instead of finding the real one each cycle and the limited-search parallel state VD based on scarce state transition [SST][8].

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 path metric and the accurate one[9]. On the other hand the SST based scheme requires predecoding and re encoding process 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. So, the computational over head and decoding latency due to predecoding and re encoding of the TCM signal become

*MD. Javeed is pursuing his master of technology in VLSI systems in Bomma institute of technology and science , Jawaharlal Nehru technological university, India. (E-mail: javeed.rahmanee@gmail.com)*

*B. Sri lakshmi is currently working as an assi.prof. in electronics and communication engineering in Jawaharlal Nehru technological university, India.( E-mail: boddu.srilakshmi@gmail.com)*

high. An add-compare select unit (ACSU) architecture based on precomputation for VDs incorporating T-Algorithm [9], which efficiently improves the clock speed of a VD with T-Algorithm for a rate ¾ code. Now, we further analyze the precomputation algorithm. A systematic way to determine the optimal precomputation steps is shown, where the minimum number of steps for critical path to achieve the theoretical iteration bound is calculated and the computational complexity overhead due to precomputation is evaluated. Then, we discuss a complete low-power VD design for the rate ¾ convolutional code[2]. Finally ASIC implementation results of VD with convolutional encoding are shown.

In this paper section II gives Information about VDs .Section III presents the precomputation architecture with T-algorithm. Design example with the modifications of survivor path memory unit(SMU) are discussed In section IV. Synthesis and power estimation results are shown in section V.

## II. VITERBI DECODER

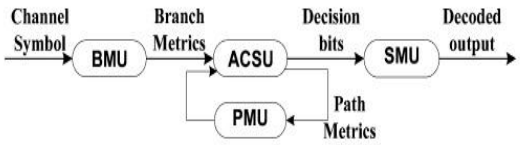


Fig.1 Diagram for viterbi decoder

A general diagram for a viterbi decoder is shown in fig. 1. First , branch metrics are calculated in the B 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, Bs are fed into the ACSU that recursively compute the path metrics (Ps) 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 Ps of the current iteration are stored in the path metric unit (PMU).

For calculating the optimal Ps and puncturing states T-Algorithm requires extra computation in the ACSU loop. Therefore, a straight forward 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 path metric.

## III. PRECOMPUTATION ARCHITECTURE

### A. Precomputation Algorithm

The Basic idea of the precomputation algorithm was presented in [9]. The Branch metric can be calculated by two types: Hamming distance and Euclidean distance [10]. Consider a VD for a convoluional code with a constraint length k, where each state receives p candidate paths. First, we expand Ps at the current time slot n(Ps(n)) as a function of Ps(n-1)to form a look-ahead computation of the optimal P-Popt (n). If branch metrics are calculated based on the Euclidean distance, popt(n) is the minimum value of Ps(n) can be get as

$$p_{opt}(n) = \min\{p_0(n), p_1(n), \ldots\ldots p_{2^{-1}k}(n)\}$$

$$= \min\{\min[p_{0,0}(n-1)+B_{0,0}(n), p_{0,1}(n-1)+B_{0,1}(n)\ldots\ldots\ldots, p_{0,P}(n-1)+B_{0,P}(n)],$$

$$\min[p_{1,0}(n-1)+B_{1,0}(n), p_{1,1}(n-1)+B_{1,1}(n), \ldots\ldots, p_{1,P}(n-1)+B_{1,P}(n)], \ldots\ldots\ldots,$$

$$\min[p_{2^{k-1}-1,0}(n-1)+B_{2^{k-1}-1,0}(n), P_{2^{k-1}-1,1}(n-1)+B_{2^{k-1}-1,1}(n), \ldots\ldots, P_{2^{k-1}-1,P}(n-1)+B_{2^{k-1}-1,P}(n)]\}$$

$$= \min\{P_{0,0}(n-1)+B_{0,0}(n),$$

$$P_{0,1}(n-1)+B_{0,1}(n), \ldots\ldots\ldots,$$

$$P_{0,P}(n-1)+B_{0,P}(n),$$

$$P_{1,0}(n-1)+B_{1,0}(n),$$

$$P_{1,1}(n-1)+B_{1,1}(n), \ldots\ldots,$$

$$P_{1,P}(n-1)+B_{1,P}(n), \ldots\ldots\ldots,$$

$$P_{2^{k-1}-1,0}(n-1)+B_{2^{k-1}-1,0}(n),$$

$$P_{2^{k-1}-1,1}(n-1)+B_{2^{k-1}-1,1}(n), \ldots\ldots,$$

$$P_{2^{k-1}-1,P}(n-1)+B_{2^{k-1}-1,P}(n)\}. \quad (1)$$

Now, 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 Bs. Thus (1) can be rewritten as

$$P_{opt} = \min\{\min(Ps(n-1) \text{ in cluster } 1)$$

$$+\min(Bs(n) \text{ for cluster } 1),$$

Min(Ps(n-1) in cluster 2)

+min(Bs(n) for cluster 2), …… ,

Min(Ps(n-1) in cluster m)

+min(Bs(n) for cluster m)}.

The minimum (Bs) for each cluster can be easily obtained from the BMU or TMU and min(Ps) at time n-1 in each cluster can be precalculated at the same time when the ACSU is updating the new Ps for time n. Theoretically, when we continuously decompose Ps(n-1), Ps(n-2),……, the precomputation scheme can be extended to Q steps. Where q is any positive integer that is less than n. Hence Popt(n) can be calculated directly from Ps(n-q) in q cycles.

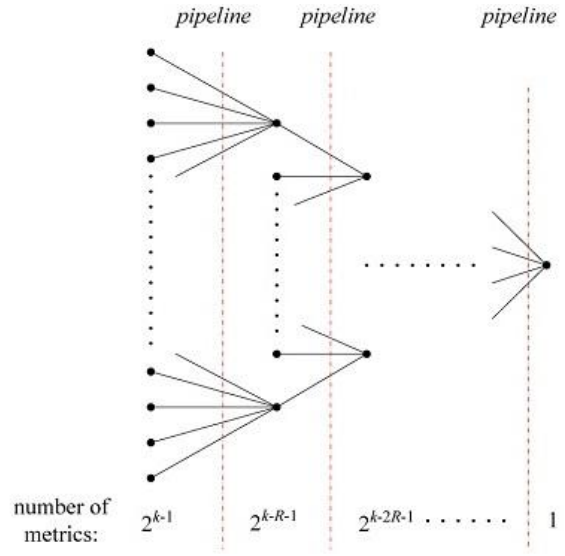


Fig.2: Precomputation Pipelinig

### B. Choosing Precomputation steps

In [9], through a design example that, q -step pre-computation can be pipelined into q stages, where the logic delay of each stage is continuously reduced as q increases. As a result, the de-coding speed of the low-power VD is greatly improved. However, after reaching a certain number of steps, qb, 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,……, so that in (1), p=2R and the logic delay of the ACSU is $T_{ACSU}=T_{addder}+T_{p\text{-}in\_comp}$, where Tadder is the logic delay of the adder to compute Ps of each candidate path that reaches the same state and Tp-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 TACSU because there will be another two input comparator in the loop to compare the new Ps with a threshold value obtained from the optimal Path metric and preset T as shown in (3)

$$T_{bound}=T_{adder}+T_{p\_in\_comp}+T_{2\text{-}in\_comp.} \quad (3)$$

To achieve the iteration bound expressed in(3), for the precomputation in each pipelining stage, we limit the comparison to be among only p 0r 2p metrics. To simplify our evaluation , we assume that each stage reduces the number of the metrics to 1/p(or2-R) of its input metrics meeting the theoretical iteration bound should satisfy (2R)qb ≥ 2k-1. Therefore qb≥ (k-1)/R and qb is expressed as (4), with a ceiling function.

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

In the design example shown in[9], with a coding rate of ¾ 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 Bs. Usually, the extra delay can be absorbed by an optimized architecture or circuit design. Even if the extra delay is 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.

Computational overhead (compared with conventional T-algorithm) is an important factor that should be carefully evaluated. Most of the computational overhead comes from adding Bs 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)

$$N_{overhead}=2^0+2^{(k-1)/q}+2^{2(k-1)/q}\ldots\ldots+2^{(q-1)(k-1)/q}$$

$$=2^{q.(k-1)/q}-1/2^{(k-1)/q}-1$$

$$=2^{k-1}-1/2^{(k-1)/q}-1 \qquad (5)$$

The estimated computational overhead according to (5) is 63/ (26/q-1) when k=7 and q.≤ 6, which 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,…..), because it usually needs more than two steps to effectively reduce the critical path(in that case, R=1 in(4) and qb 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 a small.

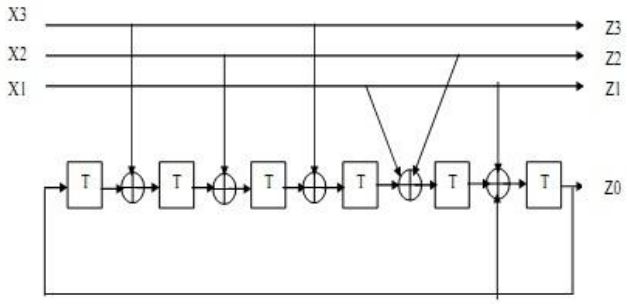


Fig. 3 Rate 3/4 convolutional encoder.

## IV. LOW POWER VITERBI DECODER DESIGN

We use the 4-D 8PSK TCM system described in[2] as the example. The rate ¾ convolutional code employed in the TCM system is shown in Fig. 3. Preliminary BER performance and architecture design for the ACSU unit have been shown in [9]. In his section, we further address the SMU design issue. Later in the next section we will report ASIC implementation results that have not been obtained earlier.

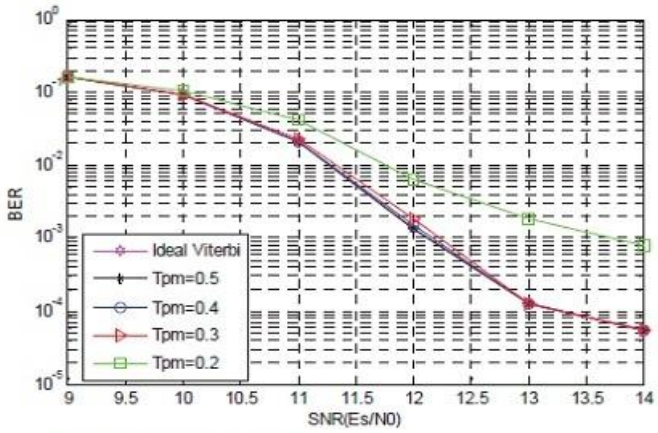


Fig.4 BER performance of T-algorithm

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 [11]. The overall coding rate is 11/12 after due to other uncoded bits in TCM system. Compared with over ideal viterbi algorithm, the threshold "Tpm" can be lowered to 0.3 with less than 0.1 dB of performance loss, while the computational complexity could be reduced by upto 90% [9] ideally. Since the performance is the same as that of the conventional T-algorithm.

### A. One step precomputation

For the convenience of our discussion we define the left most register in Fig. 3 as the most significant bit (MSB) and right most register as the least significant bit (LSB). The 64 states and path metrics are labeled from 0 to 63.

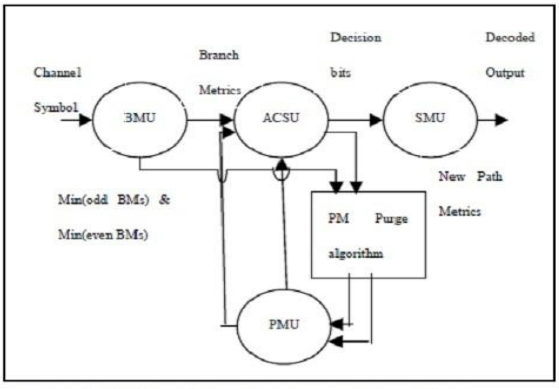


Fig. 5 Viterbi decoder with 1 step precomputation T-algorithm

A careful study reveals that the 64 states can be partitioned into two groups odd numbered Ps( when 'LSB' is 1) And even numbered (when 'LSB' is 0) The odd PMs are all extended by odd Bs (when Z0 is '1') and the even PMs are all extended by even Bs (when Z0 is '0'). The minimum P becomes:

$P_{opt}$ (n) = min {min (even Ps (n-1)) +

min(even Bs(n)), min (odd Ps (n-1))

+min(odd Bs(n)) }.

The functional diagram of the 1-step pre-computation scheme is shown in Fig. 5. In general (Path metric purge algorithm) PPAU have to wait for the new Ps from the ACSU to calculate the optimal Path metric [12], while in Fig. 5 the optimal Path metric is calculated directly from Ps in the previous cycles at the same time when the ACSU is calculating the new Ps. The details of the PPAU are shown in Fig. 6.

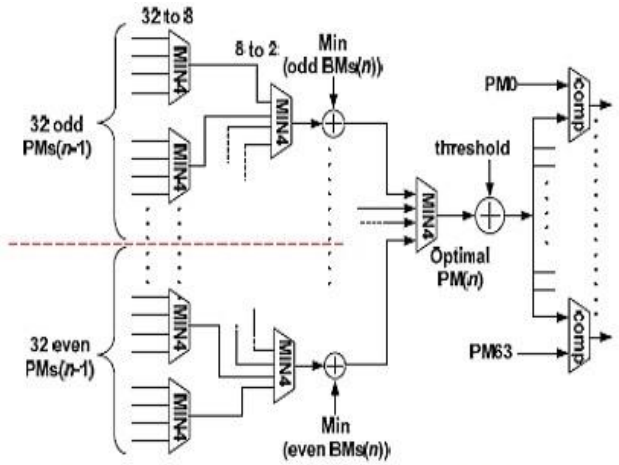


Fig. 6 Implementation of 1-step pre-computation T-algorithm

The critical path of the 1-step pre-computation scheme is

$$T_{1\text{-step-pre-T}} = 2T_{Adder} + 2T_{4\text{-in\_comp}} + 3T_{2\text{-in-comp}} \quad (6)$$

The hardware overhead of the 1-step pre-computation scheme is about 4 adders, which is negligible. Compared with the SEPC-T algorithm, however, the critical path of the 1-sept pre-computation scheme is still long[12]. In order to further shorten the critical path, we explore the 2-step pre-computation design next.

### B. Two step precomputation

#### a. Acsu design

We again need to analyze the trellis transition of the original code. In the 1-step pre-computation architecture, we have pointed out that for the particular code shown in Fig. 3, odd-numbered states are extended by odd Bs, while even-numbered states are extended by even Bs. Furthermore, the even states all extend to states with higher indices (the MSB in Fig. 3 is '1') in the trellis transition, while the odd states extend to states with lower indices (the MSB is '0' in Fig. 3). This information allows us to obtain the 2-step pre-computation data path. This process is straightforward, although the mathematical details are tedious. For clarity, we only provide the main conclusion here.

The states are further grouped into 4 clusters as described by (7). The BMs are categorized in the same way and are described by (8).

cluster3 = {Pm | 0≤m≤ 63, m mod 4 = 3}

cluster2 = {Pm | 0 ≤m≤63, m mod 4 = 1}

cluster1 = {Pm | 0 ≤m≤ 63, m mod 4 = 2}

cluster0 = {Pm | 0 ≤m≤ 63, m mod 4 = 0} (7)

BMG3 = {Bm | 0≤m≤ 15, m mod 4 = 3}

BMG2 = {Bm | 0≤m≤ 15, m mod 4 = 1}

BMG1 = {Bm | 0≤m≤ 15, m mod 4 = 2}

BMG0 = {Bm | 0≤m≤ 15, m mod 4 = 0} (8)

The optimal PM at time n is calculated as

Popt (n) = min [min {min (cluster0 (n-2))+ min

(BMG0 (n-1)), min (cluster1 (n-2))+

min (BMG1 (n-1)), min (cluster2 (n-

2)) + min (BMG3 (n-1)), min (cluster3 (n-2))+ min (BMG2(n-1)) }+ min (even Bs(n)),

min {min (cluster0 (n-2))+ min (BMG1(n-1)),

min (cluster1 (n-2))+ min (BMG0(n-1)),

min (cluster2 (n-2)) + min (BMG2 (n-1)),

min (cluster3 (n-2))+ min (BMG3(n-1))

}+ min (odd Bs(n)) (9)

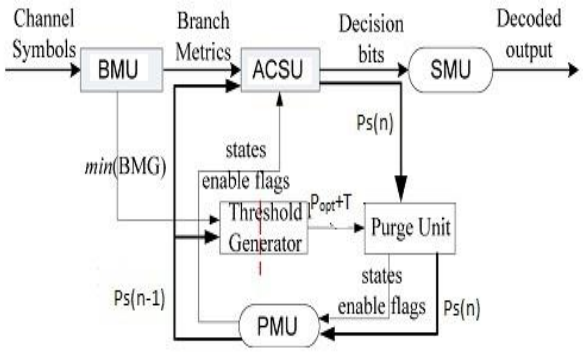


Fig. 7 two precomputation T-algorithm.

The functional block diagram of viterbi decoder with two step precomputation T-algorithm is shown in fig. 7 The minimum value of each branch metric group (BMG) can be calculated in BMU or TMU and then passed to threshold generator unit (TGU) to calculate($P_{opt}$+T)- ($P_{opt}$+T) and the new Ps are compared in the "purge unit". The architecture of TGU is shown in fig. 8 which implements the key functions of two stem precomputation scheme. In figure 8 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]. Com-pared 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)

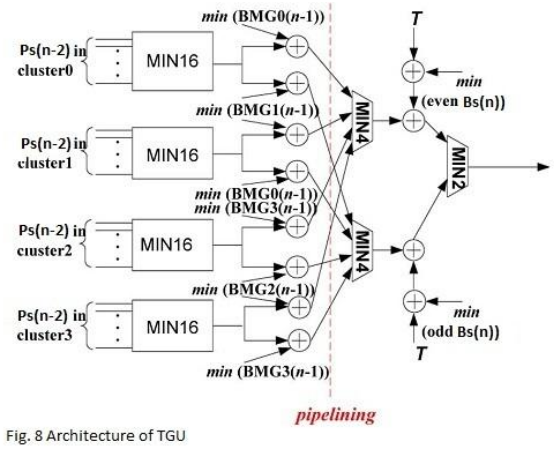


Fig. 8 Architecture of TGU

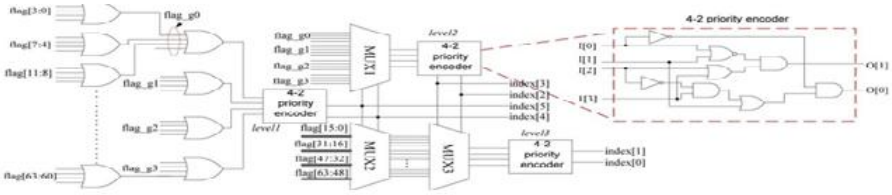


Fig. 8 64-to6 priority encoder

### 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 out-puts 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 in-corporated 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 out-putting 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 Popt ), which is always enabled, to output or trace back data. The process of searching for Popt can find out the index of the optimal state as a byproduct. However, when the estimated Popt is used [8], or in our case Popt is calculated from PMs at the previous time slot, it is difficult to find the index of the optimal state.

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 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 "index [5:4]". Then MUX2 selects one group of flags based on "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 "index [3:2]". Again, "index [3:2]" selects four flags (MUX3) as the input of the priority en-coder at level 3. Finally, the last encoder will determine "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 (10) and (11)

$$O[0] = \overline{I[0]} \cdot (I[1] + I[3]\overline{I[2]} \cdot \overline{I[1]}) = \overline{I[0]} \cdot (I[1] + I[3] \cdot \overline{I[2]}); \quad (10)$$

$$O[1] = \overline{I[0]} \cdot \overline{I[1]} \cdot (I[2] + \overline{I[2]}I[3]) = \overline{I[0] + I[1]} \cdot (I[2] + I[3]). \quad (11)$$

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 |

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(mm2) |
|---|---|---|
| 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 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 pro-posed 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 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 de-code 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.

## VII. REFERENCES

[1] F. Chan and D. Haccoun, "Adaptive viterbi decoding of convolutional codes over memory less channels," IEEE Trans. Commun. , vol. no. 45,

[2] 11, pp. 1389–1400, Nov. 1997. "Bandwidth- efficient modulations," Consultative Committee For Space Data System, Matera, Italy, CCSDS 401(3.3.6) Green Book, Issue 1, Apr. 2003.

[3] 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.

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

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

[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, M ay 2006, pp. 4811–4814.

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

[10] K. S. Arunlal and Dr. S. A. Hariprasad" An efficient viterbi decoder" International Journal of Advanced Information Technology (IJAIT) Vol. 2, No.1, February 2012

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

[12]. A.A. Peshattiwar & Tejaswini G. Panse "High Speed ACSU Architecture for Viterbi Decoder Using T-Algorithm" International Journal of Electrical and Electronics Engineering (IJEEE) ISSN (PRINT): 2231 – 5284, Vol-1, Iss-3, 2012