Hardware Efficient Low-Latency Architecture for High Throughput Rate Viterbi Decoders

Chao Cheng, *Student Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—By optimizing the number of look-ahead steps of the first layer of the previous low-latency architectures for M-step look-ahead high-throughput rate Viterbi decoders, this paper improves the hardware efficiency by large percentage with slight increase or even further decrease of the latency for the add-compare-select (ACS) computation. This is true especially when the encoder constraint length (K) is large. For example, when $\mathrm{K} = 7$ and M varies from 21 to 84, 20.83% to 41.27% of the hardware cost in previous low latency Viterbi method can be saved with only up to 12% increase or 4% decrease of the latency of the conventional M-step look-ahead viterbi decoder. The proposed architecture also relaxes the constraint on the look-ahead level M to be a multiple of K as was needed in the previous work. For example, when $\mathrm{K} = 7$ and M (indivisible by K) varies from 40 to 80, 60.27% to 69.3% latency of conventional M-step look ahead Viterbi architecture can be reduced at the expense of 148.62% to 320.20% extra hardware complexity.**

***Index Terms*—Add-Compare-Select (ACS), high-throughput rate Viterbi decoder, look-ahead implementation, low latency viterbi decoder.**

I. Introduction

VITERBI algorithm is an efficient decoding algorithm for decoding convolution codes. It is widely used in communication systems. M-step look-ahead technique can efficiently combine M trellis steps of Viterbi decoder into one trellis step and is thus frequently used for high-throughput Viterbi applications [1]–[8].

Conventional M-step look-ahead methods [1]–[7] have the problem of long latency, which usually increases linearly with M. For high-speed communication systems operating at speed in the range of Gb/s, M is usually large.

For the first time, parallel trellis paths of length-K coverage is exploited in [8] to present a novel low latency approach, which combines K trellis steps in the first layer into M/K subtrellis steps and then combines subtrellis steps into a tree structure. This K-nested layered M-step look-ahead method can efficiently reduce latency. However, its drawback is the high hardware complexity, especially when K is large. Furthermore, M is constrained to be a multiple of K and its latency in many cases can be further lowered.

In this paper, we combine the hardware efficiency of conventional and the low latency of the K-nested layered M-step Viterbi decoding method to develop a new hardware efficient method for combining the M trellis steps. The look-ahead steps in the first layer are increased to K1 ($\mathrm{K} \leq \mathrm{K1} \leq \mathrm{M}$). Since we have parallel paths when $\mathrm{K1} \geq \mathrm{K}$, we can carry out ACS-precomputation. We will show that the conventional and K-nested layered Viterbi methods are special cases of the proposed method for $\mathrm{K1} = \mathrm{M}$ and $\mathrm{K1} = \mathrm{K}$, respectively.

This paper is organized as follows. The conventional and the K-nested layered M-step Viterbi methods are reviewed and analyzed in Section II. The proposed hardware efficient low latency Viterbi method is discussed through analysis of 4 examples in Section III. Additional case studies and analysis are presented in Section IV.

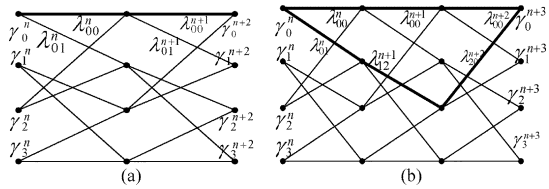

Fig. 1. Trellis diagram (a) when $\mathrm{K} = 3$ and $\mathrm{M} = 2$ and (b) when $\mathrm{K} = 3$ and $\mathrm{M} = 3$.

II. A Review on Previous M-Step Look-Ahead Viterbi Method

We present a review of previously proposed M-step look-ahead Viterbi Decoders: Conventional [1]–[7] and K-nested layered low latency M-step look-ahead Viterbi decoder [8].

A. Conventional M-Step Look-Ahead Viterbi Decoder

When $\mathrm{M} = 2$, $\mathrm{K} = 3$, the trellis structure can be represented as Fig. 1(a). In this example, the number of states is $2^{K-1} = 4$. λ_{ij}^{n} denotes the branch metric.

For example, $\hat{\lambda}_{00}^{n} = \lambda_{00}^{n} + \lambda_{00}^{n+1}$ and $\hat{\lambda}_{01}^{n} = \lambda_{00}^{n} + \lambda_{01}^{n+1}$. Since $\mathrm{M} < \mathrm{K}$, we can't have precomputation of ACS. The number of required 2-input adders and compare-select (CS) units in typical design can be given by $2^{K-1} \times 2^{K-1}$ and $2^{K-1} \times (2^{K-1} - 1)$, respectively.

When $\mathrm{M} = 3$, $\mathrm{K} = 3$, the trellis structure can be represented as Fig. 1(b). Since $\mathrm{M} = \mathrm{K}$, we can have precomputation of ACS with 2 parallel paths. For example, we have $\hat{\lambda}_{00}^{n} = \max\{\lambda_{00}^{n} + \lambda_{00}^{n+1} + \lambda_{00}^{n+2}, \lambda_{01}^{n} + \lambda_{12}^{n+1} + \lambda_{20}^{n+2}\}$. The number of required 2-input adders and CS units can be given as $2^{K-1} \times (2^2 + 2^K)$ and $2^{K-1} \times 2^{K-1}$, respectively. Similar hardware diagram corresponding to Fig. 1(b) can be found in Fig. 3 of [8]. The latency is $\mathrm{K} = 3$ clock cycles.

When $\mathrm{M} = 4$ and $\mathrm{K} = 3$, the trellis structure can be represented as shown in Fig. 2. Since $\mathrm{M} = 4$ and $\mathrm{K} = 3$, we can

Manuscript received December 18, 2007; revised May 17, 2008. Current version published December 12, 2008. This paper was recommended by Associate Editor W. X. Zheng.

The authors are with Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: chaocheng@ieee.org; parhi@umn.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/TCSII.2008.2008061

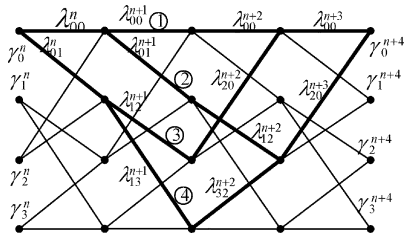

Fig. 2. Trellis diagram when $\mathrm{K} = 3$ and $\mathrm{M} = 4$.

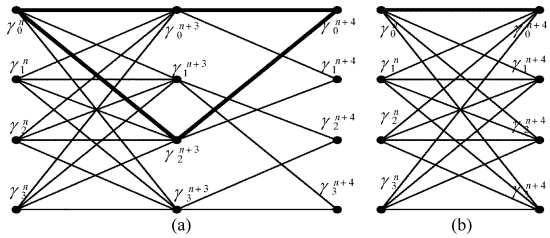

Fig. 3. Conventional Trellis diagram when $\mathrm{K} = 3$ and $\mathrm{M} = 4$, (a) with 3-step look-ahead first; (b) 4-step look-ahead.

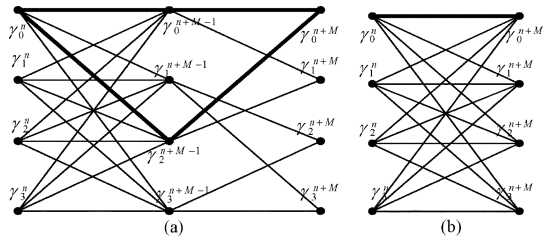

Fig. 4. Conventional Trellis diagram when $\mathrm{K} = 3$ and $\mathrm{M} > 3$, (a) with (M-1)-step look-ahead first; (b) M-step look-ahead.

have precomputation of ACS with $2^{M-K+1} = 4$ parallel paths shown in Fig. 2 as ① − ④. For example, we have

$$\begin{aligned}\hat{\lambda}_{00}^{n} &= \max\{\lambda_{00}^{n} + \lambda_{00}^{n+1} + \lambda_{00}^{n+2} + \lambda_{00}^{n+3}, \lambda_{01}^{n} + \lambda_{12}^{n+1} \\ &\quad + \lambda_{20}^{n+2} + \lambda_{00}^{n+3}, \lambda_{00}^{n} + \lambda_{01}^{n+1} + \lambda_{12}^{n+2} + \lambda_{20}^{n+3}, \\ &\quad \lambda_{01}^{n} + \lambda_{13}^{n+1} + \lambda_{32}^{n+2} + \lambda_{20}^{n+3}\} \\ &= \max\{\max\left(\lambda_{00}^{n} + \lambda_{00}^{n+1} + \lambda_{00}^{n+2}, \lambda_{01}^{n} + \lambda_{12}^{n+1} + \lambda_{20}^{n+2}\right) \\ &\quad + \lambda_{00}^{n+3}, \max\left(\lambda_{00}^{n} + \lambda_{01}^{n+1} + \lambda_{12}^{n+2}, \right. \\ &\quad \left. \lambda_{01}^{n} + \lambda_{13}^{n+1} + \lambda_{32}^{n+2}\right) + \lambda_{20}^{n+3}\}.\end{aligned}$$

The above equation can be illustrated by the bold parallel paths ① − ④ in Fig. 2, which can thus be represented as Fig. 3.

The number of required 2-input adders and CS units can be given as $2^{K-1} \times (2^2 + 2^K) + 2^{K-1} \times 2^{K-1} \times 2$ and $2^{K-1} \times 2^{K-1} + 2^{K-1} \times 2^{K-1}$, respectively. The latency is $\mathrm{K} + 2 = 5$ clock cycles.

By iteratively applying the above process, we can get the conventional Trellis diagram when $\mathrm{K} = 3$ and $\mathrm{M} > 3$ as shown in Fig. 4.

The number of required 2-input adders and comparators can be given as $2^{K-1} \times 4 \times (2^{K-2} - 1) + 2^{K-1} \times 2^{K} \times (M - K + 1)$ and $2^{K-1} \times 2^{K-1} \times (M - K + 1)$, respectively. The latency is $K + 2(M - K) = 2M - K$ clock cycles.

B. Low-Latency M-Step Look-Ahead Viterbi Decoder

K-Nested layered look-ahead method is proposed in [8]. This method assumes M is a multiple of K. For example, when $\mathrm{K} = 3$ and $\mathrm{M} = 12$, trellis diagram is shown in Fig. 2 of [8]. We can see

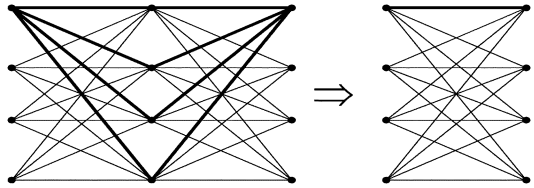

Fig. 5. Parallel path combination process after layer 2 in [8].

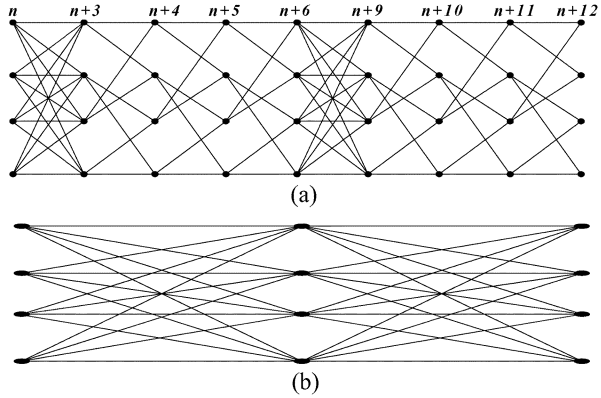

Fig. 6. Proposed trellis diagram for $\mathrm{K} = 3$, $\mathrm{M} = 12$ and $\mathrm{K1} = 6$; (a)–(b): layer 1–2.

TABLE I
SUMMARY FOR THE CASE OF $\mathrm{K} = 3$ AND $\mathrm{M} = 12$

	Proposed (K1=6)	Conventional	[8]
Latency	12	21	9
2-input adders	528	496	592

that, starting from layer 2, the parallel path combination process can be represented as shown in Fig. 5.

The number of required 2-input adders and comparators in this process can be given as $2^{K-1} \times 2^{K-1} \times 2^{K-1}$ and $2^{K-1} \times 2^{K-1} \times (2^{K-1} - 1)$, respectively; and the total hardware cost for ACS precomputation is given as

$$C_{Kong} = C_{add} + C_{CS\_r2} + C_{Cs\_rx} \times (2^{K-1} - 1)$$

where

$$C_{\mathrm{add}} = 2^{K-1} \times \left[\frac{4M}{K} \times (2^{K-1} - 1) + (2^{K-1})^2 \times \left(\frac{M}{K} - 1\right)\right]$$

$$C_{CS\_r2} = (2^{K-1})^2 \times \frac{M}{K}$$

$$C_{CS\_rx} = (2^{K-1})^2 \times \left(\frac{M}{K} - 1\right).$$

Compared with the conventional ACS-precomputation trellis for M-step look-ahead Viterbi decoders as shown in Fig. 4, which has the hardware cost increase in the order of $O((2^{K-1})^2) = O(4^K)$, that of [8] increases in the order of $O((2^{K-1})^3) = O(8^K)$. This is also the reason why the hardware cost increases fast when K is large.

We can also see that the first layer of [8] is the same as the conventional K-step look-ahead architecture as shown in Fig. 1(b). The hardware cost of the first layer thus increases in the order of $O((2^{K-1})^2) = O(4^K)$. Since increasing the look-ahead steps of the first layer will decrease the number of parallel combination process blocks as shown in Fig. 5, the total hardware cost

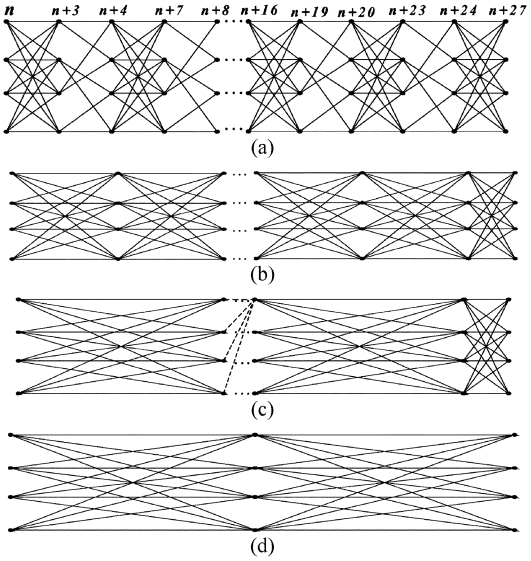

Fig. 7. Proposed trellis diagram for $\mathrm{K} = 3$, $\mathrm{M} = 27$ and $\mathrm{K1} = 4$; (a)–(d): layer 1–4.

TABLE II
SUMMARY FOR THE CASE OF $\mathrm{K} = 3$ AND $\mathrm{M} = 27$

	Proposed (K1=4)	Conventional	[8]
Latency	14	51	15
2-input adders	1408	1216	1472

can thus be controlled and the number of layers can be potentially reduced. Although increasing the look-ahead steps in the first layer will increase the latency, the final latency is also affected by the number of layers.

III. PROPOSED HARDWARE EFFICIENT LOW-LATENCY M-STEP LOOK-AHEAD VITERBI DECODER

The existence of parallel paths is required for the precomputation of ACS. When the look-ahead step is a number larger than K, there are definitely parallel paths, because the number of parallel paths is equal to 2^{M-K+1}. Since the look-ahead step of the first layer is always K in [8], we use K1 to represent that of the proposed design in this paper where $\mathrm{K} \leq \mathrm{K1} < \mathrm{M}$. The proposed design is illustrated with 4 examples in this section.

Example 1: $\mathrm{K} = 3$, $\mathrm{M} = 12$ and $\mathrm{K1} = 6$.

The proposed trellis diagram is shown in Fig. 6 and the hardware complexity and latency are summarized in Table I. From Table I, we can see that the proposed design can save 64 adders. Although latency increases by 3 clock cycles from the previous K-nested layered method, it is still far away from its conventional counterpart.

$\mathrm{M} = 27$ does not need to be a multiple of K1.

Example 2: $\mathrm{K} = 3$, $\mathrm{M} = 27$ and $\mathrm{K1} = 4$.

The proposed trellis diagram is shown in Fig. 7 and the performance characteristics of the proposed architecture are summarized in Table II.

From Table II, we can see that we can save 64 adders of previous design in [8]. Furthermore, latency is further reduced by 1 clock cycle.

M does not need to be multiples of K either.

Example 3: $\mathrm{K} = 3$, $\mathrm{M} = 11$ and $\mathrm{K1} = 4$.

The proposed trellis diagram is shown in Fig. 8.

Increasing K1 will potentially increase the latency and reduce the total hardware cost. This is illustrated in ***Example 4***.

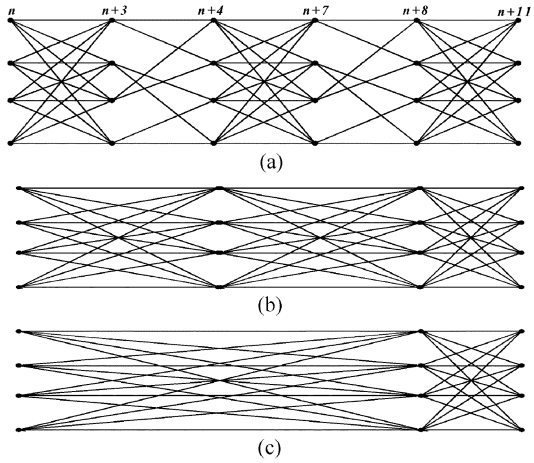

Fig. 8. Proposed trellis diagram for $\mathrm{K} = 3$, $\mathrm{M} = 11$ and $\mathrm{K1} = 4$; (a)–(c): layer 1–3.

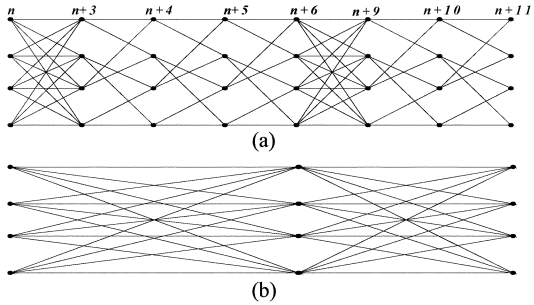

Fig. 9. Proposed trellis diagram for $\mathrm{K} = 3$, $\mathrm{M} = 11$ and $\mathrm{K1} = 6$; (a)–(b): layer 1–2.

TABLE III
SUMMARY FOR THE CASE OF $\mathrm{K} = 3$ AND $\mathrm{M} = 11$

	Proposed (K1=4)	Proposed (K1=6)	Conventional
Latency	11	12	19
2-input adders	512	480	448

Example 4: $\mathrm{K} = 3$, $\mathrm{M} = 11$ and $\mathrm{K1} = 6$.

The proposed trellis diagram is shown in Fig. 9. Performance characteristics for architectures in ***Examples 3 and 4*** are summarized in Table III.

In the proposed M-step look-ahead method, the first layer is composed of $\lfloor M/K1 \rfloor$ K1-step and one mod(M,K1)-step conventional look-ahead blocks. The later layers include $\lceil M/K1 \rceil - 1$ parallel combination blocks (Fig. 5). The total number of layers is $1 + \lceil \log_2(\lceil M/K1 \rceil) \rceil$.

From ***Examples 1*** *through* ***4***, we can see that the proposed low latency M-step look-ahead Viterbi architecture is a combination of conventional ($\mathrm{K1} = \mathrm{M}$) and K-nested layered ($\mathrm{K1} = \mathrm{K}$) method. When K1 increases from K to M, latency increases but hardware complexity is reduced. The best K1 can be searched based on system requirement on latency and hardware cost. Additional case studies are presented in the next section.

The hardware implementation of the proposed low latency Viterbi decoder is similar to that of [8]. We take ***Example 1***. $\mathrm{K} = 3$, $\mathrm{M} = 12$ with variable K1 as an example. The first layer ACS precomputation can be implemented as shown in Fig. 10. We can see that the critical path is reduced as the computation time of an adder or a CS after we apply pipelining. Considering the fact that a CS unit can be implemented by an adder and a

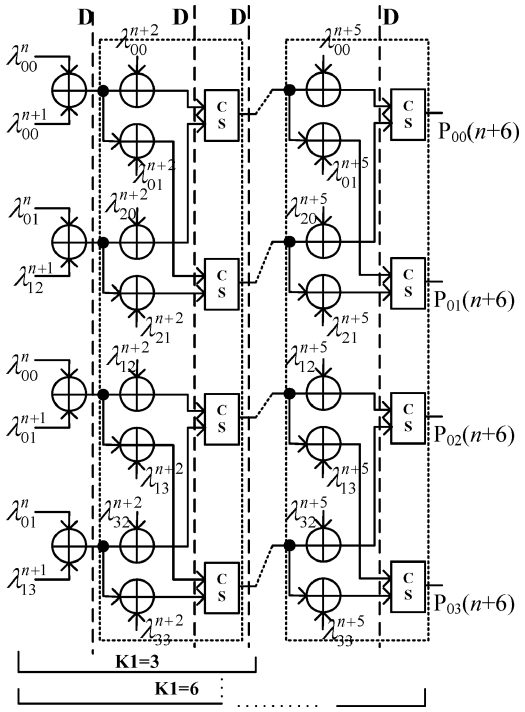

Fig. 10. Hardware Architecture for computing first layer trellis paths from state-0 in stage n to state-0-3 in stage $n+6$ with $\mathrm{M}=12$ and $\mathrm{K1}=6$.

TABLE IV
COMPARISON OF LATENCY AND COMPLEXITY FOR ACS PRECOMPUTATION WHEN $\mathrm{K}=3$

M (K=3)	Latency			Complexity (#adder)		
	Proposed (Over Conv.)	[8] (Over Conv.)	Conv.	Proposed (Saved)	[8]	Conv.
3 (K1=3)	3(100.00%)	3(100.00%)	3	64(0.00%)	64	64
6 (K1=3)	6(66.67%)	6(66.67%)	9	240(0.00%)	240	208
9 (K1=4)	10(66.67%)	9(60.00%)	15	384(7.69%)	416	352
12 (K1=6)	12(57.14%)	9(42.86%)	21	528(10.81%)	592	496
15 (K1=4)	11(40.74%)	12(44.44%)	27	736(4.17%)	768	640
18 (K1=5)	13(39.39%)	12(36.36%)	33	880(6.78%)	944	784
21 (K1=6)	15(38.46%)	12(30.77%)	39	1024(8.57%)	1120	928
24 (K1=6)	15(33.33%)	12(26.67%)	45	1168(9.88%)	1296	1072
27 (K1=4)	14(27.45%)	15(29.41%)	51	1408(4.35%)	1472	1216
30 (K1=5)	16(28.07%)	15(26.32%)	57	1520(7.77%)	1648	1360
33 (K1=5)	16(25.40%)	15(23.81%)	63	1696(7.02%)	1824	1504
36 (K1=6)	18(26.09%)	15(21.74%)	69	1808(9.60%)	2000	1648
39 (K1=5)	16(21.33%)	15(20.00%)	75	2016(7.35%)	2176	1792
42 (K1=6)	18(22.22%)	15(18.52%)	81	2128(9.52%)	2352	1936
45 (K1=6)	18(20.69%)	15(17.24%)	87	2304(8.86%)	2528	2080
48 (K1=6)	18(19.35%)	15(16.13%)	93	2448(9.47%)	2704	2224

selector and the computation time of a selector is small, we can approximate the critical path of first layer ACS precomputation as the computation time of an adder.

Note, the hardware diagrams of the second and later layers are not covered in this paper because similar structures have been discussed in detail in previous K-nested layered M-step look-ahead Viterbi method [8]. The survivor path management is similar to that in [8], and is not addressed in this paper.

IV. COMPARISON AND ANALYSIS

The hardware complexity and latency of the proposed low latency M-step look-ahead Viterbi method is given in (1) and (2), respectively. $\mathrm{C}_{add-1st}$ and C_{cs-1st} are the number of adders and comparators in the first layers, respectively. $\mathrm{C}_{add-2-last}$ and $\mathrm{C}_{cs-2-last}$ are the number of adders and comparators in the second to last layers, respectively. L_{proposed} is the latency for the proposed design.

TABLE V
COMPARISON OF LATENCY AND COMPLEXITY FOR ACS PRECOMPUTATION WHEN $\mathrm{K}=4$

M (K=4)	Latency			Complexity (#adder)		
	Proposed (Over Conv.)	[8] (Over Conv.)	Conv.	Proposed (Saved)	[8]	Conv.
4 (K1=4)	4(100.00%)	4(100.00%)	4	288(0.00%)	288	288
8 (K1=4)	8(66.67%)	8(66.67%)	12	1536(0.00%)	1536	1056
12 (K1=6)	12(60.00%)	12(60.00%)	20	2304(17.24%)	2784	1824
16 (K1=8)	16(57.14%)	12(42.86%)	28	3072(23.81%)	4032	2592
20 (K1=5)	14(38.89%)	16(44.44%)	36	4800(9.09%)	5280	3360
24 (K1=6)	16(36.36%)	16(36.36%)	44	5568(14.71%)	6528	4128
28 (K1=7)	18(34.62%)	16(30.77%)	52	6336(18.52%)	7776	4896
32 (K1=8)	20(33.33%)	16(26.67%)	60	7104(21.28%)	9024	5664
36 (K1=6)	20(29.41%)	20(29.41%)	68	8832(14.02%)	10272	6432
40 (K1=5)	18(23.68%)	20(26.32%)	76	10560(8.33%)	11520	7200
44 (K1=5)	22(26.19%)	20(23.81%)	84	11808(7.52%)	12768	7968
48 (K1=6)	20(21.74%)	20(21.74%)	92	12096(13.7%)	14016	8736

TABLE VI
COMPARISON OF LATENCY AND COMPLEXITY FOR ACS PRECOMPUTATION WHEN $\mathrm{K}=5$

M (K=5)	Latency			Complexity (#adder)		
	Proposed (Over Conv.)	[8] (Over Conv.)	Conv.	Proposed (Saved)	[8]	Conv.
5 (K1=5)	5(100.00%)	5(100.00%)	5	1216(0.00%)	1216	1216
10 (K1=5)	10(66.67%)	10(66.67%)	15	10368(0.00%)	10368	5056
15 (K1=7)	16(64.00%)	15(60.00%)	25	14208(27.21%)	19520	8896
20 (K1=9)	22(62.86%)	15(42.86%)	35	18048(37.05%)	28672	12736
25 (K1=6)	19(42.22%)	20(44.44%)	45	32512(14.04%)	37824	16576
30 (K1=7)	23(41.82%)	20(36.36%)	55	36352(22.62%)	46976	20416
35 (K1=8)	27(41.54%)	20(30.77%)	65	40192(28.39%)	56128	24256
40 (K1=9)	31(41.33%)	20(26.67%)	75	44032(32.55%)	65280	28096
45 (K1=7)	30(35.29%)	25(29.41%)	85	58496(21.41%)	74432	31936
50 (K1=6)	26(27.37%)	25(26.32%)	95	72960(12.71%)	83584	35776
55 (K1=6)	29(27.62%)	25(23.81%)	105	82112(11.46%)	92736	39616
60 (K1=7)	32(27.83%)	25(21.74%)	115	80640(20.85%)	101888	43456

$$C_{\mathrm{Proposed}} = C_{\mathrm{add\_1st}} + C_{\mathrm{add\_2\_last}} + C_{CS\_1st} + C_{CS\_2\_last} \tag{1}$$

where

$$C_{add\_1st} = \begin{cases} 2^{K-1}\times\left[4\times(2^{K-1}-1)+2^{K-1}\times 2\times(K1-K)\right] \\ \quad \times M/K1, \\ \quad when\ \mathrm{mod}(M,K1)=0 \\ 2^{K-1}\times\left[4\times(2^{K-1}-1)+2^{K-1}\times 2\times(K1-K)\right] \\ \quad \times\lfloor M/K1\rfloor+2^{K-1} \\ \quad \times\left[4\times(2^{K-1}-1)+2^{K-1}\times 2\right. \\ \qquad \left.\times(\mathrm{mod}(M,K1)-K)\right], \\ \quad when\ \mathrm{mod}(M,K1)\geq K \end{cases}$$

$$C_{add\_2\_last} = 2^{K-1}\times(2^{K-1})^2\times(\lceil M/K1\rceil-1),$$

$$C_{CS\_2\_last} = (2^{K-1})^2\times(\lceil M/K1\rceil-1)\times(2^{K-1}-1)$$

$$C_{CS\_1st} = \begin{cases} (2^{K-1})^2\times(K1-K+1)\times M/K1, \\ \quad when\ \mathrm{mod}(M,K1)=0 \\ (2^{K-1})^2\times(K1-K+1)\times\lfloor M/K1\rfloor+(2^{K-1})^2 \\ \quad \times\left[\mathrm{mod}(M,K1)-K+1\right], \\ \quad when\ \mathrm{mod}(M,K1)\geq K \end{cases}$$

$$L_{proposed} = K+2\times(K1-K)+K\times\lceil\log_2(M/K1)\rceil \tag{2}$$

TABLE VII
COMPARISON OF LATENCY AND COMPLEXITY FOR ACS PRECOMPUTATION WHEN $\mathrm{K} = 7$

M (K=7)	Latency			Complexity (#adder)		
	Proposed (Over Conv.)	[8] (Over Conv.)	Conv.	Proposed (Saved)	[8]	Conv.
7 (K1=7)	7(100.00%)	7(100.00%)	7	20224(0.00%)	20224	20224
14 (K1=7)	14(66.67%)	14(66.67%)	21	560640(0.00%)	560640	106240
21 (K1=10)	22(62.86%)	21(60.00%)	35	646656(41.27%)	1101056	192256
28 (K1=9)	27(55.10%)	21(42.86%)	49	1187072(27.68%)	1641472	278272
35 (K1=9)	25(39.68%)	28(44.44%)	63	1727488(20.83%)	2181888	364288
42 (K1=11)	29(37.66%)	28(36.36%)	77	1813504(33.38%)	2722304	450304
49 (K1=12)	33(36.26%)	28(30.77%)	91	1899520(41.78%)	3262720	536320
56 (K1=14)	35(33.33%)	28(26.67%)	105	1985536(47.79%)	3803136	622336
63 (K1=9)	32(26.89%)	35(29.41%)	119	3434752(20.92%)	4343552	708352
70 (K1=10)	34(25.56%)	35(26.32%)	133	3520768(27.91%)	4883968	794368
77 (K1=10)	34(23.13%)	35(23.81%)	147	4061184(25.13%)	5424384	880384
84 (K1=11)	36(22.36%)	35(21.74%)	161	4147200(30.47%)	5964800	966400

TABLE VIII
COMPARISON OF LATENCY AND COMPLEXITY FOR ACS PRECOMPUTATION WHEN $\mathrm{K} = 7$ FOR VARIOUS M INDIVISIBLE BY K

M (K=7)	Latency		Complexity (#adder)	
	Proposed(Over Conv.)	Conv.	Proposed (Saved)	Conv.
40 (K1=14)	29(39.73%)	73	1788928(420.20%)	425728
50 (K1=13)	33(35.48%)	93	1911808(348.48%)	548608
60 (K1=15)	37(32.74%)	113	2034688(303.01%)	671488
80 (K1=20)	47(30.72%)	153	2280448(248.62%)	917248

Based on (1) and (2) and the analysis of previous M-step look-ahead Viterbi methods in Section II, We carry out several case studies, which are summarized in Table IVthrough Table VIII.

When M is not a multiple of K, the proposed design is still efficient, although the K-nested layered look-ahead method in [8] can't be applied. This has been shown in ***Example 3 and 4*** for $\mathrm{K} = 3$. The efficiency of the proposed low latency M-step Viterbi method is shown in Table VIII for $\mathrm{K} = 7$ with various M indivisible by K. Note that the proposed design can have even less latency for certain M and K because of the reduced layers, as in example, $\mathrm{M} = 15, \mathrm{K1} = 4$ and $\mathrm{M} = 27, \mathrm{K1} = 4$ in Table IV.

From the above analysis, we can see the flexibility of the proposed M-step look-ahead Viterbi method. Hardware and latency can be balanced easily by choosing different values for K1 as shown in Fig. 11. Best K1 can be found by looking for the lowest adder ratio for a required latency. For example, in [8], design of a 4-state 10 Gb/s SERDES requires a latency of less than 60 ns. For an implementation with 0.13-um technology, 48-stage ACS precomputation with previous low latency design has a latency of $15 \times 1.5 = 22.5$ ns. The proposed design has latency of $18 \times 1.5 = 27$ ns and can still satisfy the latency requirement. However, our design saves 9.47% hardware complexity as shown in the last row of Table IV.

V. CONCLUSION

Low latency M-step look-ahead Viterbi decoder for high throughput rate implementation is important for many applications. This paper optimizes the hardware efficiency and latency of previous low latency K-nested layered Viterbi method by increasing the look-ahead steps in the first layer. The remaining layers are combined with the same scheme used in [8]. The results from extensive case studies show that proposed design is efficient in terms of both hardware complexity and latency. Furthermore, the limitation on the look-ahead steps M (to be a multiple of K) is relaxed.

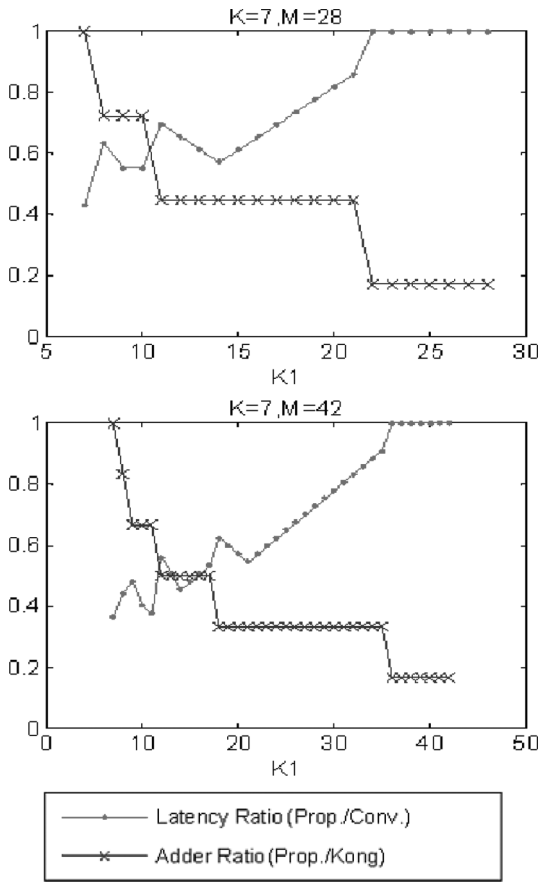

Fig. 11. Search process for best K1 when $\mathrm{K} = 7$ and $\mathrm{M} = 28/42$.

REFERENCES

[1] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS-bottleneck," *IEEE Trans. Commun.*, vol. 37, no. 8, pp. 785–790, Aug. 1989.

[2] G. Fettweis and H. Meyr, "High-rate Viterbi processor: A systolic array solution," *IEEE J. Sel. Areas. Commun.*, vol. 8, pp. 1520–1543, Oct. 1990.

[3] G. Fettweis and H. Meyr, "High-speed parallel Viterbi decoding: Algorithm and VLSI-architecture," *IEEE Commun. Mag.*, pp. 46–55, May 1991.

[4] T. Gemmeke, M. Gansen, and T. G. Noll, "Implementation of scalable power and area efficient high throughput Viterbi decoders," *IEEE J. Solid-State Circuits*, vol. 37, no. 7, pp. 941–948, Jul. 2002.

[5] V. S. Gierenz, O. Weis, T. G. Noll, I. Carew, J. Ashley, and R. Karabed, "A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-μm CMOS Viterbi decoder," in *Proc. 2000 IEEE Int. Conf. Application-Specific Syst., Architectures, and Processors*, pp. 195–201.

[6] K. K. Parhi, "An improved pipelined MSB-first add-compare-select unit structure for Viterbi decoders," *IEEE Trans. Circuits Syst., I: Reg. Papers*, vol. 51, no. 3, pp. 504–511, March 2004.

[7] P. J. Black and T. H. Meng, "A 140-Mb/s, 32-state, radix-4 Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992.

[8] J. Kong and K. K. Parhi, "Low-latency architectures for high-throughput Viterbi decoders," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 12, no. 6, pp. 642–651, Jun. 2004.