High-Throughput Turbo Decoder with Parallel Architecture for LTE Wireless Communication Standards

by

Rahul Shrestha, Roy Paily

in

IEEE Transactions on Circuits and Systems I: Regular Papers

Report No: IIIT/TR/2014/-1

Centre for VLSI and Embedded Systems Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2014
High-Throughput Turbo Decoder With Parallel Architecture for LTE Wireless Communication Standards

Rahul Shrestha, Graduate Student Member, IEEE, and Roy P. Paily, Member, IEEE

Abstract—This work focuses on the VLSI design aspect of high-speed maximum a posteriori (MAP) probability decoder which are intrinsic building-blocks of parallel turbo decoders. For the logarithmic-Bahl–Cocke–Jelinek–Raviv (LBCJR) algorithm used in MAP decoders, we have presented an ungrouped backward recursion technique for the computation of backward state metrics. Unlike the conventional decoder architectures, MAP decoder based on this technique can be extensively pipelined and retimed to achieve higher clock frequency. Additionally, the state metric normalization technique employed in the design of an add-compare-select-unit (ACSU) has reduced critical path delay of our decoder architecture. We have designed and implemented turbo decoders with 8 and 64 parallel MAP decoders in 90nm CMOS technology. VLSI implementation of an 8 × parallel turbo-decoder has achieved a maximum throughput of 439 Mbps with 0.11 nJ/bit/iteration. Similarly, 64 × parallel turbo-decoder has achieved a maximum throughput of 3.3 Gbps with an energy-efficiency of 0.079 nJ/bit/iteration. These high-throughput decoders meet peak data-rates of 3GPP-LTE and LTE-Advanced standards.

Index Terms—Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm, maximum a posteriori (MAP) decoder, parallel turbo decoding and VLSI design, turbo codes, wireless communications, 3GPP-LTE/LTE-advanced.

I. INTRODUCTION

WITH the advent of powerful smart phones and tablets, wireless multimedia communication has become an integral part of our life. In the year 2012, approximately 700 million such gadgets were estimated to be sold worldwide, and their requirement of data rate has been on an exponential trajectory [1]. This has led to the deployment of new standards which can support higher data rate. The Third-Generation-Partnership-Project (3GPP) conceived an air-interface termed as 3GPP-LTE (3GPP-long-term-evolution) release-8 and was reformed to 3GPP-LTE release-9 which supports a peak data rate of 326.4 Mbps [2]. 3GPP-LTE-Advanced standard has presented in [8]. Similarly, parallel architecture of turbo decoder with contention-free interleavers, memories and interconnecting networks. Maximum achievable throughput of such decoder with $P \times \text{radix-2^m}$ MAP decoders for a block length of $N$ and a sliding window size of $M$ is given as

$$\Theta_T = \frac{P \times \omega \times F}{2 \times \rho} \times \frac{Z \times M / \omega}{(Z + 2) \times M / \omega + \delta_{\text{map}} + \delta_{\text{ext}} + \delta_{\text{dec}}} \quad (1)$$

where $Z = N / M$, $F$ is a maximum operating clock frequency, $\rho$ represents a number of decoding iterations, $\delta_{\text{map}}$ is a pipeline delay for accessing data from memories to MAP decoders, $\delta_{\text{ext}}$ is a pipeline delay for writing extrinsic information to memories, and $\delta_{\text{dec}}$ is a decoding delay of MAP decoder [7]. This expression suggests that the achievable throughput of parallel turbo decoder has dominant dependencies on number of MAP decoders, operating clock frequency and number of decoding iterations. Thereby, valuable contributions have been reported to improve these factors. An implementation of parallel turbo decoder which uses retimed and unified radix-2$^m$ MAP decoders, for Mobile WiMAX (wireless-interoperability-for-microwave-access) and 3GPP-LTE standards, is presented in [8]. Similarly, parallel architecture of turbo decoder with contention-free interleaver is designed for higher throughput applications in [9]. For 3GPP-LTE standard, reconfigurable and parallel architecture of turbo decoder with a novel multistage interconnecting networks is implemented in [10]. Recently, a peak data rate of 3GPP-LTE standard has been achieved by parallel turbo decoder implemented in [11]. Subsequently, a processing schedule for the parallel turbo decoder has been proposed to achieve 100% operating efficiency in [7]. In [12], high throughput parallel turbo decoder based on the algebraic-geometric properties of quadratic-permutation-polynomial (QPP) interleaver has been proposed. An architecture incorporating a stack of $16 \times$ MAP decoders with an optimized state-metric initialization scheme for low decoder latency and use of multi-antenna techniques and support for relay nodes in the LTE-Advanced air-interface have made its new releases capable of supporting peak data rate(s) of 3 Gbps milestone [4]. For reliable and error-free communication in these recent standards, turbo code has been extensively used because it delivers near-optimal bit-error-rate (BER) performance [5]. However, the iterative nature of turbo decoding imposes adverse effect which defers turbo decoder from achieving high-throughput benchmarks of the latest wireless communication standards. On the other hand, extensive research on the parallel architectures of turbo decoder has shown its promising capability to achieve higher throughput, albeit, at the cost of large silicon-area [6]. Parallel turbo decoder contains multiple maximum a posteriori (MAP) probability decoders, contention-free interleavers, memories and interconnecting networks. Maximum achievable throughput of such decoder with $P \times \text{radix-2^m}$ MAP decoders for a block length of $N$ and a sliding window size of $M$ is given as

$$\Theta_T = \frac{P \times \omega \times F}{2 \times \rho} \times \frac{Z \times M / \omega}{(Z + 2) \times M / \omega + \delta_{\text{map}} + \delta_{\text{ext}} + \delta_{\text{dec}}} \quad (1)$$
high throughput is presented in [13]. Another contribution which includes a very high throughput parallel turbo decoder for LTE-Advanced base station applications is presented in [14]. Recently, a novel hybrid decoder-architecture of turbo low-density-parity-check (LDPC) codes for multiple wireless communication standards has been proposed in [15].

Based on the comprehensive overview of recent standards for wireless communication, primary motive of our research is to conceive an architecture of turbo decoder for high throughput application. We have focused on an improvement of maximum clock frequency \(F\) which eventually improves an achievable throughput of parallel turbo decoder from (1). Works with similar motivations have been reported in the literature [16]–[18]. So far, no work has reported parallel turbo decoder that can achieve higher throughput beyond 3 Gbps milestone targeted for the future releases of 3GPP-LTE-Advanced. The contributions of our work presented in this paper are summarized as follows.

1) We propose a modified MAP-decoder architecture based on a new ungrouped backward recursion scheme for the sliding window technique of the logarithmic-Bahl–Cocke–Jelinek–Raviv (LBCJR) algorithm and a new state metric normalization technique. The suggested techniques have made provisions for re-timing and deep-pipelining in the architectures of the state-metric-computation-unit (SMCU) and MAP decoder, respectively, to speed up the decoding process.

2) As a proof of concept, an implementation in 90 nm CMOS technology is carried out for the parallel turbo decoder with \(8 \times \text{radix-2 MAP-decoders} which are integrated with memories via pipelined interconnecting networks based on contention-free QPP interleavers. It is capable of decoding 188 different block lengths ranging from 40 to 6144 with a code-rate of \(1/3\) and achieves more than the peak data rate of 3GPP-LTE. We have also carried out synthesis-study and postlayout simulation of a parallel turbo decoder with \(64 \times \text{radix-2 MAP-decoders} which can achieve the milestone throughput of 3GPP-LTE-Advanced.

3) Subsequently, the fixed-point simulation for BER performance analysis of parallel turbo decoder is carried out for various iterations, quantization and code rates.

4) Finally, the key characteristics of parallel turbo decoder presented in this work are compared with the reported contributions from the literature.

II. THEORETICAL BACKGROUND

Transmitter and receiver sections of a wireless gadget which supports 3GPP-LTE/LTE-Advanced standards are shown in Fig. 1(a). Each of these sections has three major parts: digital-baseband module, analog-RF module, and multiple-input-multi-output (MIMO) antennas. In the digital-baseband module of transmitter, sequence of information bits \(U_k \quad \forall k = \{1, 2, 3, \ldots, N\}\) is processed by various submodules and is fed to the channel encoder. It generates a systematic bit \(x_{sk}\), parity bit \(x_{pk}\) and \(x_{pk}\) for each information bit using convolutional-encoders (CEs) and I (QPP-interleaver). These encoded bits are further processed by remaining submodules; finally, the transmitted digital data from baseband are converted into quadrature and inphase analog signals by a digital-analog-converter (DAC). Analog signals, those are fed to the multiple analog-RF modules, are up-converted to a RF frequency, amplified, bandpassed and transmitted via MIMO antennas which transform RF signals into electromagnetic waves for the transmission through wireless channel, as shown in Fig. 1(a). At the receiver, RF signals provided by multiple antennas to analog-RF modules are band-pass filtered to extract signals of the desired band, then they are low-noise-amplified and down-converted into baseband signals. Subsequently, these signals are sampled by analog-digital-converter (ADC) of the digital-baseband module, where various submodules process such samples and are fed to the soft-demodulator. It generates a \(a\) priori logarithmic-likelihood-ratios (LLRs) \(\lambda_{a}\), \(\lambda_{pk}\) and \(\lambda_{pk}\) for the transmitted systematic and parity bits, respectively, and are fed to turbo decoder via serial-parallel converter. Turbo decoders work on graph based approach and are parallel concatenation of MAP decoders, as shown in Fig. 1(a). Basically, the MAP decoder uses BCJR algorithm to process input \(a\) priori LLRs and then determine the values of \(a\) posteriori LLRs for the transmitted bits. Extrinsic information values are computed as \(\lambda_{a1k} = \{\lambda_{a1k} + L_{1k}(U_{k}) - \lambda_{a2k}\}\) and \(\lambda_{a2k} = \{\lambda_{a1k} + L_{2k}(U_{k}) - \lambda_{a1k}\}\) where \(L_{1k}(U_{k})\) and \(L_{2k}(U_{k})\) are a \(a\) posteriori LLRs from MAP decoders; \(\lambda_{a2k}\) and \(\lambda_{a1k}\) are de-interleaved and interleaved values, respectively, of the extrinsic information. As shown in Fig. 1(a), the values of extrinsic information are iteratively processed by MAP decoders to achieve near-optimal BER performance. Finally, a
posteriori LLR values, those are generated by turbo decoder, are processed by rest of the baseband submodules. Ultimately, a sequence of decoded bits $V_k$ is obtained, as shown in Fig. 1(a).

On the other hand, conventional BCJR algorithm for MAP decoding includes mathematically complex computations. It delivers near-optimal error-rate performance at the cost of huge memory and computationally intense VLSI (very-large-scale-integration) architecture, which imposes large decoding delay [19]. These shortcomings have made this algorithm inappropriate for practical implementation. Logarithmic transformations of miscellaneous mathematical equations involved in BCJR algorithm have scaled down the computational complexity as well as simplified its architecture from an implementation perspective [20], and such procedure is referred as logarithmic-BCJR (LBCJR) algorithm. Furthermore, huge memory requirement and large decoding delay can be controlled by employing sliding window technique for LBCJR algorithm [21]. This is a trellis-graph based decoding process in which $N$ stages are used for determining a posteriori LLRs $\mathcal{L}_k(U_k) \forall k = \{1, 2, 3, \ldots, N\}$ where each stage comprises of $N_t$ trellis states. LBCJR algorithm traverses forward and backward of this graph to compute forward $\alpha_k(s_i)$ as well as backward $\beta_k(s_i)$ state metrics, respectively, for each trellis state such that $k \in N$ and $i \in N_s$. As shown in Fig. 1(b), for states $s_0$ and $s_1$, forward and backward state metrics during their respective traces are computed as

$$
\alpha_k(s_0) = \max\{\alpha_k \cdot 1(s'_0) + \gamma_k(s'_0, s_0), \alpha_k \cdot 1(s'_1) + \gamma_k(s'_1, s_1)\}
$$

$$
\beta_k(s_1) = \max\{\beta_k + 1(s'_0) + \gamma_k(s'_0, s_1), \beta_k + 1(s'_1) + \gamma_k(s'_1, s_1)\}
$$

respectively, where $\max$ is a logarithmic approximation which simplifies mathematical computations of BCJR algorithm. Based on max-log-MAP approximation, this function operates as $\max(A, B) = \max(A, B)$. Similarly, log-MAP approximation computes $\max$ as $\max(A, B) = \max(A, B) + \ln(1 + e^{-A-B})$ [20]. Similarly, for an arbitrary state transition from $s'_i$ to $s_j$ such that $(i, j) \in N_s$, $\gamma_k(s'_i, s_j)$ is a branch metric which uses a priori LLRs for the computation and is expressed as

$$
\gamma_k(s'_i, s_j) = 1/2 \cdot [U_k \cdot L_{a_k} + L_c]
$$

$$
\cdot \{x_{a_k} \cdot \lambda_{a_k} + x_{p1k} \cdot \lambda_{p1k} + x_{p2k} \cdot \lambda_{p2k}\}
$$

where $L_{a_k}$ accounts for a priori information which is interleaved/de-interleaved extrinsic information value in turbo decoding. In addition, $L_c$ represents channel reliability measure and is approximated as $L_c \approx 2$ when the value of fading amplitude is $a = 1$ [20]. $A$ posteriori LLR value of a trellis stage is computed after the computation of all state and branch metrics. Assuming that $s$ represents trellis transition where $s^t(\delta)$ and $s^m(\delta)$ corresponds to start and end states, the $a$ posteriori LLR value for $k$th trellis stage is computed as [20]

$$
L_k(U_k) = \max_{k, i} \gamma_k(s'_i, s_j) \Rightarrow f(\delta) = \max_{k, i} \{f(\delta)\}
$$

where the function $f(\delta)$ is expressed as

$$
f(\delta) = \alpha_{k-1} \{s^t(\delta) + \gamma_k(\delta) + \beta_k \{s^m(\delta)\}. \}
$$

Additionally, $\delta : \{s'_i, s_j \Rightarrow U_k = 0/1$ indicates set of all trellis transitions when the information bit is “0” or “1.” " Basically, the sliding window technique for LBCJR (SW-LBCJR) algorithm segregates $N$ trellis stages into $N/M$ windows where each window comprises of $M$ trellis stages and has a processing time of $T_{sw}$ [21]. Fig. 1(c) shows time-scheduling of SW-LBCJR algorithm for various operations those are carried out in successive sliding windows (SWs). In the first time-slot $T_{sw}$, branch metrics of the first SW (SW1) are computed. Subsequently, branch metrics for SW2 as well as dummy-backward-recursion that estimates boundary backward state metrics for SW1 are accomplished in the time-interval $2T_{sw} < t < 2T_{sw}$. Similarly, effective-backward-recursion for SW1 is initiated during the interval $2T_{sw} < t < 3T_{sw}$ where the computation of a posteriori LLRs for SW1 begins simultaneously. Other operations such as dummy-backward-recursion and forward-recursion runs in parallel during this interval. Moreover, such process is carried out successively for all the SWs as shown in Fig. 1(c). Thereby, conventional SW-LBCJR algorithm has a decoding delay of $2 \times T_{sw}$. It needs to store branch metrics for two SWs and forward state metrics for one SW [22].

III. PROPOSED TECHNIQUES

We now present the suggested techniques for sliding window approach and state metric normalization of LBCJR algorithm.

A. Modified Sliding Window Approach

This approach for LBCJR algorithm is based on an ungrouped backward recursion technique. Unlike the conventional SW-LBCJR algorithm, this technique performs backward recursion for each trellis stage, independently, for the computation of backward state metrics. For a sliding window size of $M$, such an ungrouped backward recursion for $k$th stage begins from $(k + M - 1)$th stage in the trellis graph. Each of these backward recursions is initiated with logarithmic-equiprobable values assigned to all the backward state metrics of $(k + M - 1)$th trellis stage as

$$
\beta_{k+M-1}(s_j) = \ln\left(\frac{1}{N_s}\right) \forall j \in N_s.
$$

Simultaneously, the branch metrics are computed for successive trellis stages and are used for determining the state metric values using (2). After computing $N_s$ backward state metrics of $k$th trellis stage by an ungrouped backward recursion, all the forward state metrics of $(k - 1)$th trellis stage are computed. It is to be noted that the forward recursion starts with an initialization at $k = 0$ such that

$$
\alpha_{k-0}(s_i) = 0 \quad \forall i \in 0 \text{ and } \alpha_{k-0}(s_i) = -\infty \quad \forall i \neq 0.
$$

Thereafter, a posteriori LLR value of $k$th trellis stage is computed using the branch metrics of all state transitions, as well as forward and backward state metrics from $(k - 1)$th and $k$th trellis stages, respectively, as given in (4). Paralleling such ungrouped backward recursions for successive trellis stages in order to compute their a posteriori LLRs is a primary concern of our work. For the sake of clarity, we have used handful of new notations while explaining this approach for LBCJR algorithm. For example, $B_k$ and $A_k$ represent sets of $N_s$ backward and forward state metrics, respectively, of $k$th trellis stage. They are expressed as $B_k = \{\beta_k(s_i) \mid i \in N_s^0, 0 \leq i < N_s\}$ and $A_k = \{\alpha_k(s_i) \mid i \in N_s^0, 0 < i < N_s\}$ where $N_s^0$ is a set of natural numbers including zero. Similarly, a set of all branch metrics, associated with the transitions from $(k - 1)$th to $k$th trellis stages, is denoted by $\Gamma_k$ which is expressed as
\[ \Gamma_k = \{ \gamma_k(\chi) | \chi : \text{a set of all state transitions} \}. \]

Since, there are multiple ungrouped backward recursions in this approach, we have denoted \( B_k \) for different ungrouped backward recursions as \( \{ B_k \}^u \) such that \( u \in U \) and \( U \) is a set of all ungrouped backward recursions for each time-interval. Fig. 2(a) illustrates the suggested ungrouped backward recursions for LBCJR algorithm with a value of \( M = 4 \). It shows the computation of backward state metrics for \( k = 1 \) and \( k = 2 \) trellis stages. First ungrouped backward recursion (denoted as \( u = 1 \)) starts with the computation of \( \{ B_k=3 \}^{u=1} \) using the initialized backward state metrics from \( k = 4 \) trellis stage. Thereafter, \( \{ B_{k-2} \}^{u=1} \) is computed using \( \{ H_{k-3} \}^{u=1} \); finally, an effective set of backward state metric \( \{ B_k=1 \}^{u=1} \), which is then used in the computation of \( a posteriori \) LLR for \( k = 1 \) trellis stage, is obtained using the value of \( \{ B_k \}^{u=1} \). Similarly, such successive process of second ungrouped backward recursion (\( u = 2 \)) is carried out to compute an effective-set of \( \{ B_{k=2} \}^{u=2} \) for \( k = 2 \) trellis stage, as shown in Fig. 2(a). In this suggested-approach, time-scheduling of various operations to be performed for the computation of successive \( a posteriori \) LLRs is schematically presented in Fig. 2(b). This scheduling is illustrated for \( M = 4 \), where the trellis stages and time intervals are plotted along y-axis and x-axis respectively. As the time progresses, a set of branch metrics (denoted as \( \Gamma_k \)) is computed in each time interval. Therefore, \( \Gamma_k \) \( \forall 1 \leq k \leq 8 \) are successively computed from the time interval \( t_1 \) to \( t_8 \), as shown in Fig. 2(b). Similarly, ungrouped backward recursions begin from \( t_k \)th time interval because the branch metrics required for these recursions are available from this interval onward. Therefore, referring Fig. 2(b), operations performed from this interval onward are systematically explained as follows:

- \( t_5 \): A first ungrouped backward recursion (denoted by \( u = 1 \)) begins with the computation of \( \{ B_{k=1} \}^{u=1} \) which uses the initialized backward state metrics from \( k = 4 \) trellis stage. Since, this backward recursion is performed to compute an effective-set of backward state metrics for \( k = 1 \), it is started from \( (k + M - 1) = 4 \)th trellis stage.

- \( t_6 \): A consecutive-set \( \{ B_{k-2} \}^{u=1} \) is computed for the continuation of first ungrouped backward recursion. Simultaneously, a second ungrouped backward recursion starts from the initialized trellis stage \( k = 5 \) with the computation of a new-set \( \{ B_{k-1} \}^{u=2} \).

- \( t_7 \): First ungrouped backward recursion ends in this interval with the computation of effective-set \( \{ B_{k=1} \}^{u=1} \) for \( k = 1 \) trellis stage. In parallel, second ungrouped backward recursion continues with the computation of consecutive-set \( \{ B_{k=3} \}^{u=2} \). Similarly, a new-set \( \{ B_{k=1} \}^{u=3} \) is computed and it marks the start of third ungrouped backward recursion. Initialization of all the forward state metrics of set \( A_{k=0} \) is also carried out, as given in (7).

- \( t_8 \): An effective-set \( \{ B_{k-2} \}^{u=2} \) is obtained with the termination of second ungrouped backward recursion and a consecutive-set \( \{ B_{k-4} \}^{u=3} \) is computed for an ongoing third ungrouped backward recursion. Simultaneously, fourth ungrouped backward recursion begins with the computation of a new-set \( \{ B_{k=6} \}^{u=4} \). Using an initialized set \( A_{k=0} \), a set of forward state metrics \( A_{k=1} \) is determined. A \( a posteriori \) LLR value \( L_{k-1}(U_k) \) of the trellis stage \( k = 1 \) is computed using forward, backward and branch metrics from the sets \( A_{k-0}, \{ B_{k=1} \}^{u=1} \) and \( \Gamma_{k=1} \) respectively.

Decoding delay \( \delta_{dec} \), for the computation of \( a posteriori \) LLRs for \( M = 4 \) is a sum of seven time-intervals \( (\delta_{dec} = \sum_{i=1}^{7} t_i) \), as shown in Fig. 2(b). Therefore, it can be concluded that the decoding delay of this approach is \( \delta_{dec} = (2 \times T_{sv}) - 1 \). It has been observed that from \( t_7 \) interval onward, three \( \{ H_k \}^u \) sets are simultaneously computed in each interval. Thereby, in general, this approach requires \( M - 1 \) units to accomplish such parallel task of ungroup backward recursion. However, implementation aspects of the MAP decoder based on this approach is discussed in Section IV.

B. State Metric Normalization Technique

Magnitudes of forward and backward state metrics grow as recursions proceed in the trellis graph. Overflow may occur without normalization, if the data widths of these metrics are finite. There are two commonly used state metric normalization techniques: subtractive and modulo normalization techniques [23]. In the subtractive normalization technique, normalized forward and backward state metrics for \( kth \) trellis stage are computed as

\[
\alpha_k(s_j)^* = [\alpha_k(s_j) - \max_{0 \leq j < N_s} \{ \alpha_{k-1}(s_j) \}], \quad \beta_k(s_j)^* = [\beta_k(s_j) - \min_{0 \leq j < N_s} \{ \beta_{k+1}(s_j) \}],
\]

respectively [23]. On the other side, two’s complement arithmetic based modulo normalization technique works with a principle that the path selection process during forward/backward recursion depends on the bounded values of path metric difference [24]. The normalization technique suggested in our work is focused to achieve high-speed performance of turbo decoder from an implementation perspective. Assume that \( s_x \) and \( s_y \) states at \((k-1)\)th stage as well as \( s'_x \) and \( s'_y \) states at \((k+1)\)th stage are associated with \( s_x \) state at \( k \)th stage in a trellis graph.
Thereby, normalization of a forward state metric at \( s_x \) state is performed as

\[
\alpha_k(s_x) = \max \left\{ F_{k+1}^1 \alpha_{k-1}(s'_1) - \{ F_{k+1}^2 \alpha_{k-1}(s'_2) \}, \quad i \in N_s \right\}
\]

where \( F_{k+1}^1 \) and \( F_{k+1}^2 \) are the path metrics for two different state-transitions: \( s'_1 \) to \( s_x \) and \( s'_2 \) to \( s_x \) respectively. They are expressed as \( F_{k+1}^1 = \{ \alpha_{k-1}(s'_1) + \gamma_k(s'_2, s_x) \} \) and \( F_{k+1}^2 = \{ \alpha_{k-1}(s'_2) + \gamma_k(s'_1, s_x) \} \). In the above (9), \( \alpha_{k-1}(s'_i) \) such that \( i \in N_s \) is a normalizing factor which is one of the previously computed forward state metrics of \( N_s \) states from \( (k-1) \)th trellis stage. Similarly, a backward state metric at \( k \)th trellis stage can be normalized as

\[
\beta_k(s_x) = \max \left\{ F_{k+1}^1 - \beta_{k+1}(s'_1), \quad \beta_{k+1}(s'_2) \right\}
\]

where the path metrics are represented as \( F_{k+1}^1 = \{ \beta_{k+1}(s'_1) + \gamma_{k}(s'_2, s_x) \} \) and \( F_{k+1}^2 = \{ \beta_{k+1}(s'_2) + \gamma_{k}(s'_1, s_x) \} \). Similarly, the normalizing factor is \( \beta_{k+1}(s'_j) \) from a state among \( N_s \) trellis states at \( (k+1) \)th stage. It is to be noted that such normalizing factors \( \alpha_{k-1}(s'_i) \) and \( \beta_{k+1}(s'_j) \) can be used for computing all \( N_s \) normalized forward and backward state metrics, respectively, at \( k \)th trellis stage.

From an implementation perspective, an ACSU (add-compare-select-unit) computes normalized state metric in MAP decoder which requires \( N_s \) such ACSUs to determine all forward/backward state metrics of a trellis stage. Fig. 3 shows the ACSU architectures based on modulo, subtractive and suggested normalization techniques. These ACSUs are used for computing a normalized forward state metric at \( s_0 \) state of a trellis graph with \( N_s = 8 \) states, as shown in Fig. 3(d). An ACSU design based on (9) is shown in Fig. 3(b). In this architecture, path metrics are subtracted with a normalizing factor \( \alpha_{k-1}(s'_i) \) using subtractors along second stage and then multiplexed to obtain a normalized forward state metric \( \alpha_k(s_0)^* \). Similarly, state-of-the-art ACSU-architecture for modulo normalization technique is presented in Fig. 3(a) and it obtains normalized forward state metric value with controlled overflow using two two-input-XOR gates [25]. However, an ACSU for subtractive normalization technique requires additional comparator circuit for \( N_s = 8 \) states to obtain a value of \( \max_{1 \leq j \leq N_s} \{ \alpha_{k-1}(s_j) \} \) from (8), as shown in Fig. 3(c). Eventually, a maximum value obtained from this comparator is subtracted with the state metric for normalization. These architectures of ACSUs are presented for max-log-MAP LBCJR algorithm for high-speed applications [20]. However, its degradation in BER performance, as compared to Log-MAP LBCJR algorithm, may be avoided by using an extrinsic scaling process [25]. Critical paths of ACSUs based on the suggested approach, modulo, and subtractive normalization techniques are highlighted in Fig. 3(a)–(c) and are quantified as

\[
\begin{align*}
\tau_{new} &= \tau_{add} + \tau_{sub} + \tau_{mux} \\
\tau_{mod} &= \tau_{add} + \tau_{sub} + \tau_{mux} + \tau_{xor} \\
\tau_{sub} &= \tau_{add} + \tau_{sub} + \tau_{mux} + \tau_{xor} + \tau_{mux}
\end{align*}
\]

respectively, where \( \tau_{add}, \tau_{sub}, \tau_{mux} \), and \( \tau_{xor} \) are the delays imposed by an adder, a subtractor, a multiplexer, and an XOR gate respectively. In this work, stack of \( N_s \) ACSUs for computing all the forward/backward state metrics is collectively referred as SMCU. We have performed a postlayout simulation study, in 90 nm CMOS technology, of SMCUs with \( N_s = 8 \) based on these state metric normalization techniques and their key characteristics obtained are presented in Table I. Subsequently, design-synthesis and static-timing-analysis are performed under worst-corner case with a supply of 0.9 V at 125 °C operating temperature. It can be seen that SMCU based on the suggested approach have 21.82% and 60.77% better operating clock frequencies than the SMCUs based on modulo and subtractive normalization techniques respectively. Suggested SMCU design consumes 17.87% lesser silicon-area than the SMCU based on subtractive normalization technique. However, it has area overhead of 6.02% in comparison with modulo normalization based SMCU. Total power consumed at 100 MHz clock frequency by this SMCU is 6% lesser and 2.13% more than subtractive and modulo normalization techniques, respectively, as shown in Table I. The suggested approach for state metric normalization technique has shown better operating clock frequency with the nominal degradations in area-occupied and power-consumed, as compared to modulo normalization technique.

### IV. Decoder Architectures and Scheduling

We next present the MAP-decoder architecture and its scheduling based on the proposed techniques. Detail discussion on
the design of high-speed MAP decoder, and its implementation trade-offs, are carried out. Furthermore, parallel architecture of turbo decoder and QPP interleaver used in this work are presented.

A. MAP-Decoder Architecture and Scheduling

Decoder architecture for LBCJR algorithm based on an ungrouped backward recursion technique is shown in Fig. 4(a). Basically, it includes five major subblocks: BMCU (branch-metric-computation-unit), ALCU (a posteriori-LLR-computation-unit), RE (registers), LUT (look-up-table), and SMCU that uses suggested state metric normalization technique. The BMCU processes $n$ a priori LLRs of systematic and parity bits ($\lambda_{nk}, \lambda_{p1k}, \ldots, \lambda_{pnn}$), where $n$ is a code-length, to successively compute all the branch metrics in each of the sets $\Gamma_k$, $1 \leq k \leq N$. A posteriori LLR for kth trellis stage is computed by ALCU using the sets of state and branch metrics, as shown in Fig. 4(a). Subblock RE is a bank of registers used for data-buffering in the decoder. LUT stores the logarithmic equiprobable values, as given in (6), for backward state metrics of $(k + M - 1)$th trellis stage which initiates an ungroup backward recursion for kth trellis stage. As discussed earlier, SMCU3 computes $N_k$ forward or backward state metrics of a trellis stage. Based on the time-scheduling that is illustrated in Fig. 2(b) from Section III, we have presented an architecture of MAP decoder for $M = 4$ in Fig. 4(a). Thereby, three $(M - 1)$ SMCUs are used for ungrouped backward recursions in this decoder architecture and are denoted as SMCU1, SMCU2 and SMCU3. Similarly, forward state metrics for successive trellis stages are computed by SMCU4. For better understanding of the decoding process, a graphical representation of data launched by different registers, those are included in the decoder architecture, for successive clock cycles are illustrated in Fig. 4(b). In this decoder architecture, input a priori LLRs as well as a priori information $I_{nk}$ for the successive trellis stages are sequentially buffered through RE1 and then processed by BMCU, which computes all the branch metrics of these stages, as shown in Fig. 4(a). These branch metric values are buffered through a series of registers and are fed to SMCUs for backward recursion, SMCU4 for forward recursion and ALCU for computation of a posteriori LLRs. In the fifth clock cycle, branch metrics of set $\Gamma_{k-4}$ are launch from RE2 and are used by SMCU1 along with the initial values of backward state metrics from LUT to compute backward state metrics of $\{B_{k-3}\}_{n=1}$, for the first ungrouped backward recursion, and then stored in RE8, as shown in Fig. 4(b). These stored values of RE8 are launched in the sixth clock cycle and are fed to SMCU2 along with a branch metric set $\Gamma_{k-3}$, from RE4, to compute a set $\{B_{k-2}\}_{n=1}$ which is then stored in RE9. In the same clock cycle, computation of $\{B_{k-4}\}_{n=2}$, for second ungrouped backward recursion, can be computed by SMCU1 using $\Gamma_{k-5}$ launched by RE2, and store them in RE8. Both these sets of backward state metrics are launched by RE8 and RE9 in the seventh clock cycle, as illustrated in Fig. 4(b). It can be seen that similar pattern of computations for branch and state metrics are carried out for successive trellis stages, referring Fig. 4(a) and (b). By using branch metric sets from RE11, SMCU4 is able to compute sets of forward state metrics $A_{nk}$ for successive trellis stages. The sets of forward state, backward state and branch metrics via RE13, RE10, and RE12, respectively, are fed to ALCU, as shown in Fig. 4(a). Thereby, a posteriori LLRs are successively generated by ALCU from the ninth clock cycle onward, for the value of $M = 4$, as shown in Fig. 4(b). Henceforth, from an implementation perspective, the decoding delay $\delta_{dec}$ of this MAP decoder is $2 \times M$ clock cycles.

B. Retimed and Deep-Pipelined Decoder Architecture

In the suggested MAP-decoder architecture, SMCU4 with buffered feedback paths is used in forward recursion and it imposes a critical path delay of $\tau_{new}$ from (11). On the other hand, SMCU4 architecture can be retimed to shorten this critical path delay. For a trellis-graph of $N_k = 4$, retimed data-flow-graph of SMCU with buffered feedback paths for computing the forward state metrics of successive trellis stages is shown in Fig. 5(a). It has four ACSUs based on suggested state metric normalization technique and they compute forward state metrics using $\alpha_{k-1}(s_1^f)$ normalizing factor. However, this retimed data-flow-graph based architecture operates with a clock $(clk2)$ that has double the frequency of clock $(clk1)$ at which the branch metrics are fed, as shown in Fig. 5(b). Otherwise, it may miss the successive forward state metrics from $(k + 1)$th stage to compute state metrics for $k$th trellis stage. It can be seen that the critical path of this SMCU has only a subtractor delay, thereby; this retimed-unit can be operated at higher clock frequency $f_{clk2}$. However, remaining units of MAP decoder such as BMCU, ALCU, and SMCUs, those are used for ungrouped backward recursions, must operate at a clock frequency of $f_{clk1} = f_{clk2}/2$. Fortunately, these units have feed-forward digital architectures which are suitable for deep-pipelining. Basically, BMCU and ALCU are combinational designs and can be pipelined with ease. An advantage of the suggested MAP decoder architecture is that the SMCUs for backward recursion process can also be pipelined. This increases a data-processing frequency $(f_{clk1})$ at which the branch metrics are fed to retimed SMCU that is already operating at higher clock frequency. However, such retimed SMCU is not suitable for conventional MAP decoder because the SMCUs for backward recursion in
such decoder-design have feedback architectures. Thereby, they cannot be pipelined to enhance the data-processing frequency, though the retimed SMCU are operating at higher clock frequency [11], [25].

1) High-Speed MAP Decoder Architecture: In this work, we have presented architecture of MAP decoder for turbo decoding, as per the specifications of 3GPP-LTE/LTE-Advanced [3]. It has been designed for an eight-state convolutional encoder with a transfer function of $\{1, (1 + D + D^3)/(1 + D^2 + D^3)\}$. The basic block-diagrams of the turbo decoder and encoder can be referred from Fig. 1(a). For $N_s = 8$ trellis graph which is devised based on this transfer function, four parent branch metrics are required in each trellis stage to compute state metrics as well as a posteriori LLR value. Based on (3), these four branch metrics are given as

$$
\gamma_k(s'_k, s_0) = -L_{ak}/2 - (\lambda_{ak} + \lambda_{fk1})
$$

$$
\gamma_k(s'_2, s_2) = -L_{ak}/2 - (\lambda_{ak} - \lambda_{fk1})
$$

$$
\gamma_k(s'_2, s_2) = L_{ak}/2 + (\lambda_{ak} - \lambda_{fk1})
$$

$$
\gamma_k(s'_7, s_7) = L_{ak}/2 + (\lambda_{ak} + \lambda_{fk1})
$$

where the channel reliability measure has a value of $L_C = 2$ in (3). BMCU architecture that computes these parent branch metrics is shown in Fig. 6. One-bit right-shifter divides a value by two; and an inverted value can be added with a decimal equivalent of one to produce a two's complement equivalent of a fixed-point number. This architecture has been pipelined with two stages of register delays along the feed-forward paths. On the other side, eight ACSUs are collectively stacked to build a feed-forward pipelined-architecture of SMCU, which can be used for ungrouped backward recursion, as shown in Fig. 6. It computes $\beta_k(s_0)$ to $\beta_k(s_7)$ values for $N_s = 8$ trellis states and are normalized with the value of $\beta_{k+1}(s'_7)$ such that $j \in N_s$. Basically, ALCU is a simple feed-forward architecture of adders, subtractors and comparators. These adders are used for computing path metric values, as given in (5), comparators determine maximum path metric values and are subtracted to produce a posteriori LLRs. Additionally, six stages of register delays are used to pipeline ALCU in this work. These individually pipelined units are included in the MAP decoder design to make it a deep-pipelined architecture, as shown in Fig. 6. In addition, a retimed architecture of SMCU based on the data-flow-graph of

Fig. 5 has been used as a RSMCU (retimed-state-metric-computation-unit) for determining the values of $N_s$ forward state metrics for successive trellis stages. Incorporating all the pipelined feed-forward units in the MAP decoder of Fig. 6, both SMCU and ALCU has a subtractor and a multiplexer in their critical paths, where as BMCU has a subtractor along this path. Thereby, the critical path delay among all these units is a sum of subtractor and multiplexer delays ($\tau_{clk1} = \tau_{sub} + \tau_{mul}$). It decides the data-processing clock frequency of $f_{clk1}$ and is proportional to the achievable throughput of decoder. Similarly, a subtractor delay $\tau_{clk}$ fixes the retimed clock frequency $f_{clk2}$ for RSMCU. Fig. 6 shows the clock distribution of MAP decoder in which $clk2$ signal for RSMCU is frequency divided, using a flip-flop, to generate $clk1$ signal that is fed to feed-forward units. Since each of the feed-forward SMCU's are single-stage pipelined with register delays, one additional stage of register bank is required to buffer branch metrics for each SMCU, as shown in Fig. 6. Thereby, the decoding delay of this MAP decoder is given as

$$
\delta_{dec} = (\eta_{bmcu} + 1) \times 2 \times M + (\eta_{bmcu} + 1) + (\eta_{alu} + 1) \times clk\ cycle
$$

(13)

where $\eta_{bmcu}$, $\eta_{alu}$, and $\eta_{alu}$ are the number of pipelined stages in SMCU, BMCU, and ALCU respectively. Subsequently, respective clock cycle delays imposed by these units are $(\eta_{bmcu} + 1)$, $(\eta_{alu} + 1)$, and $(\eta_{alu})$ in the above expression.

2) Multiclock Domain Design: In the suggested multiclock design of decoder architecture, it is essential to synchronize the signals crossing between clock domains. Fig. 7(a) shows two clock domains of high-speed MAP-decoder architecture: DPU (deep-pipelined-unit) and RSMCU. DPU includes all the feed-forward units and is operated with a clock $clk1$, and RSMCU is fed with another clock $clk2$ which has twice the clock frequency of $clk1$. In this design, set of branch metrics $\lambda_k(s)$ and set of forward state metrics $A_k(s)$ are the signals crossing from lower-to-higher and higher-to-lower clock-frequency domains respectively. Timing diagram illustrated in
signal. Since, as shown in Fig. 7(b). Finally, is a nm SMCUs for ungrouped backward recursion in this work, we have considered signal and the synchronizer, it is initiated after some delay nm signal. Similarly, the output signal signal in the next positive edge which satis

timing requirements of \( \lambda_{k'} \) signal. Therefore, \( \Gamma^*_k \) signals crossing from \( \Lambda_k \) to \( \Lambda^*_k \) domain violates setup and hold time criteria of \( \lambda_{k'2} \) signal, as indicated in the timing diagram of Fig. 7(a). Thereby, RSMCU and DPU generate undefined-values of \( \Lambda_k \) and a posteriori LLRs respectively. A promising solution to mitigate this problem is to include two-stage-synchronizers along the signal-paths crossing these clock domains [26]. Two-stage-synchronizer is basically two flip-flops connected in series and it samples an asynchronous signal to generate a version of the signal that posses transitions, synchronized to the local clock. We have included such synchronizers along the paths of \( \Gamma_k \) with \( \Lambda_{k1} \) signal and \( \Lambda_{k2} \) signal to generate synchronous signals \( \Gamma^*_k \) and \( \Lambda^*_k \), respectively, as shown in Fig. 7(b). Timing diagram shows that the first data \( \{ \text{data} \} \) of \( \Gamma_k \) is sampled by second positive edge of \( \Lambda_{k1} \) signal and the synchronizer generates \( \Gamma^*_k \) signal in the next positive edge which satisfies timing requirements of \( \lambda_{k'1} \) signal. Similarly, the output signal \( \Lambda_k \) from RSMCU at higher clock frequency are synchronized to lower frequency using a similar synchronizer which operates with \( \lambda_{k1} \) signal, as shown in Fig. 7(b). Finally, a posteriori LLRs are synchronously generated with \( \lambda_{k1} \) signal.

3) Implementation Trade-Offs: Deep-pipelined MAP-decoder architecture of our work has a lower critical path delay and is suitable for high-speed applications. However, the affected design-metric is its large silicon-area due to the requirement of \( M-1 \) SMCUs for ungrouped backward recursions. On the other hand, conventional MAP decoder requires two backward recursion SMCUs for computing dummy and effective backward state metrics [25]. Basically, the value of \( M \) must be five to seven times the constraint length \( k_{cr} \) of convolutional encoder to achieve near-optimal error-rate performance [22]. Since the convolutional encoder has a value of \( k_{cr} = 4 \) in this work, we have considered \( M = 32 \) for our decoder design. Memories required by conventional decoder to store branch and forward-state metrics are excluded in the suggested MAP-decoder architecture [25]. Thereby, it is important to find out which is more expensive in terms of hardware efficiency: \( M - 1 \) SMCUs for ungrouped backward recursions or two SMCUs for backward recursion plus memories for branch and state metrics? For the sake of fair comparison among the suggested and traditional decoder architectures, we have implemented our design in 130 nm CMOS technology with a supply of 1.2 V and the key characteristics are presented in Table II. An architecture of MAP decoder presented in [27] is based on retimed radix-4 \( \times \) 4 two-dimensional ACSU. By relocating adders and retiming the architecture of parallel radix-2 ACSUs, for concurrent operation, the critical path of this architecture includes two adders and a multiplexer. Thereby, the suggested MAP decoder operates at a higher clock frequency by 54.75% but with an area overhead of 7.55%, in comparison with the reported work in [27]. Scalable radix-4 MAP decoder architecture has been designed and implemented in [28]. It has conventional ACSU with radix-4 architecture which includes two adders and two multiplexers along its critical path. Comparatively, the MAP decoder presented in this paper operates with 76.23% better clock frequency than the reported work of [28] and has an area overhead of 39.62%, as shown in Table II. Another MAP decoder based on block-interleaved pipelining technique is presented in [18]. It has radix-2 architecture for ACSU which is pipelined to achieve a critical path delay that is equal to the sum of two adders and multiplexer delays. Thereby, the suggested decoder-architecture has shorter critical path delay as compared to the work of [18]. Irrespective of different CMOS technology nodes, the normalized design-area of the suggested decoder is approximately \( 2 \times \) lesser than the reported work of [18].

C. Parallel Turbo-Decoder Architecture

With an objective of designing a high-throughput parallel turbo decoder that meets the benchmark data-rate of 3GPP specification [3], we have used a stack of MAP decoders with multiple memories and interconnecting-networks (ICNWs). Parallel turbo decoder achieves higher throughput as it simultaneously processes \( N/P \) input \( a \text{ priori} \) LLRs in each time instant and reduces the decoding delay of every half-iteration [6]. For 188 different block lengths of

![Fig. 7](image-url)
3GPP-LTE/LTE-Advanced, one of the parallel configuration $P$, such that $P \in \{1, 2, 4, 8, 32, 64\}$, can be used for turbo decoding [3]. In this work, a parallel configuration of $P = 8$ has been used for a code-rate of 1/3, as shown in Fig. 8(a). It can be seen that the input a priori LLRs $\lambda_{sb}$, $\lambda_{p1k}$, and $\lambda_{p2k}$ are channeled into three different banks of memories. Each bank comprises of eight memories (MEM1 to MEM8) and $N/P$ a priori LLRs are stored in each of these memories. For seven-bit quantized values of a priori LLRs and a maximum value of $N = 6144$, these banks store 126 kB of data. These stored a priori LLR values are fetched in each half-iteration and are fed to the stack of 8x MAP decoders. As shown in Fig. 8(a), memory-bank for $\lambda_{sb}$ is connected with 8x MAP decoders via ICNW. Multiplexed LLR values from memory-banks of $\lambda_{p1k}$ and $\lambda_{p2k}$ are also fed to these MAP decoders. It is to be noted that the ICNW is used for an interleaving phase while turbo decoding. It processes contention free addresses generated by dedicated address-generation-units (AGUs) and then routes data outputs from memories to correct MAP decoders to avoid the risk of memory-collision [29]. In this work, we have used an area-efficient ICNW which is based on the master–slave Batcher network [11]. In addition, this ICNW has been pipelined to maintain the optimized critical path delay of MAP decoder. Fig. 8(b) shows the ICNW used in this work with nine pipelined stages. The AGUs in ICNW generate contention free pseudorandom addresses of quadratic-permutation-polynomial (QPP) interleaver based on

$$II(i) = \{ \{( f_1 \times s \times K) + (f_2 \times s^2 \times K^2) \} + (2 \times f_2 \times s \times K \times i) + \{( f_1 \times i) + (f_2 \times i^2)\} \} \mod N.$$ (14)

where $i = \{1, 2, 3, \ldots, K\}$, $K = \lceil N/P \rceil$, and $s = \{0, 1, 2, \ldots, 7\}$ for AGU0 to AGU7 respectively [10]. Similarly, $f_1$ and $f_2$ are the interleaving factors whose values are determined by the turbo block length of 3GPP standards [3]. Addresses generated by AGUs are fed to the network of master-circuits, denoted by "M" in Fig. 8(b), which generate select signals for the network of slave-circuits, denoted by "S." Data-outputs from the memory-bank are fed to slave network and are routed to 8x MAP decoders. Stack of MAP decoders and memories MEX1 to MEX8, for storing extrinsic information, are linked with ICNW. For the eight-bit quantized extrinsic information, 48 kB of memory is used in the decoder architecture. During the first half-iteration, the input a priori LLR values $\lambda_{sb}$ and $\lambda_{p2k}$ are sequentially fetched from memory-banks and are fed to 8x MAP decoders. Then, the extrinsic information produced by these MAP decoders is stored sequentially. Thereafter, these values are fetched and pseudorandomly routed to MAP decoders using ICNW and are used as a priori-probability values for the second half-iteration. Simultaneously, $\lambda_{sb}$ soft values are fed pseudorandomly via ICNW and the multiplexed $\lambda_{p2k}$ values are fed to the MAP decoders to generate a posteriori LLRs $L_{kb}(U_{k})$. This completes one full-iteration of parallel turbo decoding. Similarly, further iterations can be carried out by generating the extrinsic information and repeating the above procedure.

V. PERFORMANCE ANALYSIS, VLSI IMPLEMENTATION AND COMPARISON OF RESULTS

To achieve near-optimal error-rate performance, a priori LLR values, state and branch metrics are quantized for the simulation which evaluates BER performance delivered by the fixed-point model of parallel turbo decoder. Fig. 9 shows the BER curves obtained from the simulation of parallel turbo decoder with $P = 8$ for a low-effective code-rate of 1/3 at 5.5 and 8 full-iterations. For these magnitudes of design metrics, value of $M = 32$ is required to deliver an optimum BER performance. It can be seen that the turbo decoder with quantized values of $n_{sb} = 7$ bits, $n_{be} = 9$ bits and $n_{se} = 8$ bits for input a priori LLRs, state and branch metrics, respectively, can achieve a low BER of $10^{-5}$ at 0.6 dB, while decoding for eight full-iterations. Turbo decoder with such quantization can perform 0.5 dB better than the decoder with $\{n_{sb}, n_{be}, n_{se}\} = \{5, 8, 7\}$ bits of quantized values for eight full-iterations, as shown in Fig. 9. Similarly, BER simulation of parallel turbo decoder with quantization $\{n_{sb}, n_{be}, n_{se}\} = \{7, 9, 8\}$ bits is performed at high effective code-rate of 0.95 for different iterations, as shown in Fig. 10. It shows that an iterative decoding of parallel turbo decoder with 12 full-iterations can perform 0.6 dB better than
Fig. 9. BER performance in AWGN channel using BPSK modulation for a low effective code-rate of 1/3, \( N = 6144 \) \( (f_1 = 263, f_2 = 489) \), \( M = 32 \), \( P = 8 \) and \( \nu = 1 \). The legend format is (Iterations, No. of bits for input \textit{a priori} LLR values, No. of bits for state metrics, No. of bits for branch metrics).

Fig. 10. BER performance in AWGN channel using BPSK modulation for a high effective code-rate of 0.95, \( N = 6144 \) \( (f_1 = 263, f_2 = 489) \), \( M = 32 \), \( P = 8 \), and quantization of (7, 9, 8).

The decoder with eight full-iterations at a BER of \( 10^{-6} \). Similarly with 5.5 full-iterations, this parallel turbo decoder has BER of \( 10^{-5} \) at an \( E_b/N_0 \) value of 2.5 dB. In this work, we have confined our simulations within two extreme corners of the code-rates: low effective code rate of 1/3 and high effective code rate of 0.95. It is to be noted that for modern system, the full range of code-rates between these corners must be supported [12]. On the other hand, BER performance of turbo decoder degrades as parallelism further increases, because the sub-block length \( (N/P) \) becomes shorter. Based on the simulation carried out for fixed-point model of turbo decoder, the value of \( M \) must be approximately \( N/P \) for such highly parallel decoder-design to achieve near-optimal BER performance, while decoding for eight full-iterations. Therefore, we have chosen the values of \( M = 96 \) for our parallel turbo-decoder model with the configuration of \( P = 64 \) to achieve near-optimal error-rate performance.

We now present VLSI implementations of parallel turbo decoders with different configurations. Parallel turbo-decoder architecture with the configuration \( P = 8 \) has been implemented in 90 nm CMOS technology. Based on the simulations for BER performances, quantized values are decided and a sliding window size of \( M = 32 \) has been considered for this implementation. It can process 188 different block lengths, as per the specifications of 3GPP-LTE/LTE-Advanced, ranging from 40 to 6144 which decide the magnitudes of interleaving factors \( f_1 \) and \( f_2 \) for the AGUs of ICNW [3]. Additionally, it has a provision of decoding at 5.5 as well as 8 full-iterations. For this design, functional simulations, timing analysis and synthesis have been carried on with Verilog-Compiler-Simulator, Prime-Time, and Design-Compiler tools, respectively, from Synopsys. Subsequently, place-&-route and layout verifications are carried out with CADence-SOC-Encounter and CADence-Virtuoso tools respectively. Presences of high-speed MAP decoders and pipelined ICNWs in the parallel turbo decoder have made it possible to achieve timing closure at a clock frequency of 625 MHz. In these dual-clock domain MAP decoders, timing closures at 625 MHz and 1250 MHz have been achieved by deep-pipelined feed-forward units and RSMCU respectively. With the value of \( M = 32 \) and pipelined-stages of \( \{ \eta_{mcu}, \eta_{rmcu}, \eta_{apmcu} \} = \{1, 2, 6\} \), decoding delay of \( \delta_{dec} = 138 \) clock cycles from (13) and pipeline delay of \( \delta_{map} = \delta_{ext} = 9 \) clock cycles are imposed by MAP decoders and ICNW respectively. Thereby, throughputs achieved by an implemented parallel turbo decoder with \( P = 8 \) are 301.69 Mbps and 438.83 Mbps for 8 and 5.5 full-iterations, respectively from (1), for a low effective code-rate of 1/3. However, an achievable throughput is 201.13 Mbps for a high effective code-rate of 0.95, while decoding for 12 full-iterations to achieve near-optimal BER performance. In the suggested MAP decoder architecture, data is directly extracted between the registers and SMCMCs rather being fetched from the memories, as it is performed in the conventional sliding window technique for LBCJR algorithm [22], and this may increase the power consumption. To reduce such dynamic power dissipation of our design, fine grain clock gating technique has been used in which enable condition is incorporated with the register-transfer-level code of this design and it is automatically translated into clock gating logic by the synthesis tool [26]. The total power (dynamic plus leakage powers) consumed while decoding a block length of 6144 for eight iterations is 272.04 mW. At the same time, this design requires extra SMCMCs as well as registers and it has resulted in an area overhead which can be mitigated to some extent by scaling down the CMOS technology node. Fig. 11(a) shows the chip-layout of parallel turbo decoder...
TABLE III  
KEY CHARACTERISTICS COMPARISON OF PARALLEL TURBO DECODER IMPLEMENTATIONS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology (nm)</td>
<td>90</td>
<td>90</td>
<td>65</td>
<td>65</td>
<td>65</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>Voltage (V)</td>
<td>1.0</td>
<td>1.0</td>
<td>0.9</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>0.9</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>Max. block length</td>
<td>6144^4</td>
<td>6144^4</td>
<td>6144^4</td>
<td>6144^4</td>
<td>2400^4</td>
<td>6144^4</td>
<td>4096^4</td>
<td>6144^4</td>
<td>6144^4</td>
<td></td>
</tr>
<tr>
<td>Parallel MAP cores</td>
<td>8</td>
<td>6</td>
<td>64</td>
<td>64</td>
<td>16</td>
<td>32</td>
<td>35 PEs</td>
<td>8</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>MAP architecture</td>
<td>radix-2</td>
<td>radix-2</td>
<td>radix-2</td>
<td>radix-4</td>
<td>radix-4</td>
<td>radix-2T</td>
<td>radix-2^4</td>
<td>radix-2^2</td>
<td>radix-2^2</td>
<td></td>
</tr>
<tr>
<td>Sliding window size</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td>14:30</td>
<td>192</td>
<td>20</td>
<td>32</td>
<td>32</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Core area (mm^2)</td>
<td>6.1(3.17^4)</td>
<td>19.75(10.3^4)</td>
<td>8.3</td>
<td>2.94</td>
<td>7.7</td>
<td>4.87</td>
<td>2.1</td>
<td>9.61</td>
<td>3.57</td>
<td>(1.785^3)</td>
</tr>
<tr>
<td>Gate count</td>
<td>694 k</td>
<td>5304 k</td>
<td>5.8 M</td>
<td>1574 k</td>
<td>—</td>
<td>—</td>
<td>602 k</td>
<td>2833 k</td>
<td>553 k</td>
<td>11000 k</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>625</td>
<td>625</td>
<td>400</td>
<td>410</td>
<td>450</td>
<td>200</td>
<td>275</td>
<td>175</td>
<td>302</td>
<td>250</td>
</tr>
<tr>
<td>Throughput (Mbps)</td>
<td>301.69 (438.83^3)</td>
<td>2274 (330.7^3)</td>
<td>1280</td>
<td>1013</td>
<td>2150</td>
<td>292</td>
<td>130</td>
<td>1400</td>
<td>390.6^3</td>
<td>186</td>
</tr>
<tr>
<td>Max. no. of iterations</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>5.5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>5.5</td>
<td>8</td>
</tr>
<tr>
<td>Power (mW)</td>
<td>272.04</td>
<td>1450.5</td>
<td>845</td>
<td>956</td>
<td>—</td>
<td>183.2</td>
<td>219</td>
<td>1356</td>
<td>788.9</td>
<td></td>
</tr>
<tr>
<td>Ener. eff. (nJ/bit/iter.)</td>
<td>0.11</td>
<td>0.079</td>
<td>0.11</td>
<td>0.17</td>
<td>—</td>
<td>0.078</td>
<td>0.21</td>
<td>0.12</td>
<td>0.37 (0.12^2)</td>
<td>0.61 (0.20^2)</td>
</tr>
<tr>
<td>(n_{of}. n_{a}) (bit)</td>
<td>(7, 9)</td>
<td>(7, 9)</td>
<td>(6, 10)</td>
<td>(−, −)</td>
<td>(−, −)</td>
<td>(6, 2)</td>
<td>(6, 9)</td>
<td>(5, 8)</td>
<td>(5, 10)</td>
<td>(−, −)</td>
</tr>
<tr>
<td>(n_{of}. n_{b}) (bit)</td>
<td>(8, 10)</td>
<td>(8, 10)</td>
<td>(10, 8)</td>
<td>(−, −)</td>
<td>(9, 10)</td>
<td>(6, 4)</td>
<td>(10, 12)</td>
<td>(8, −)</td>
<td>(−, −)</td>
<td>(−, −)</td>
</tr>
</tbody>
</table>

^T: Normalization energy factor \( N.E.F. = (1.0 \ V/1.2 \ V)^2 \times \{90 \ nm/130 \ nm\} = 0.3; \ T: Normalization area factor = \{90 \ nm/130 \ nm\}^2 = 0.5; \ L: Normalization area factor = \{65 \ nm/90 \ nm\}^2 = 0.32.

^1: Postlayout simulation results; ^2: On chip measured results; §: Throughput achieved at 5.5 iterations; †: Reconfigurable parallel turbo decoder architecture. n_{a}: No. of bits for input a priori LLR values; n_{a}: No. of bits for state metrics; n_{b}: No. of bits for branch metrics; n_{c}: No. of bits for a posteriori-logarithmic-likelihood-ratio.

^\dagger: Supports 3GPP-LTE standard; ^\ddagger: Supports 3GPP-LTE-Advanced standard; ^\ast: Supports 3GPP-LTE-Advanced and WiMAX standards; ^\&: Supports 3GPP-LTE and WiMAX standards; ^\|: Supports WiMAX IEEE 802.16e, WiMAX IEEE 802.11n, DVB-RCS, HomePlug-AV, CMMB, DTMB, and 3GPP-LTE standards.

Table III summarizes the key characteristics of implemented decoders of our work and compares them with the reported parallel turbo-decoder implementations in the literature [7], [8], [10]–[15]. These contributions include on-chip measured and postlayout simulated results in 65 nm, 90 nm and 130 nm CMOS technologies. Normalized area occupation and energy efficiency are also included in Table III for fair comparison. Among the contributions in 65 nm CMOS technology, the postlayout simulation of parallel turbo decoder with \( P = 64 \) has 29% better throughput than the throughput reported in [14]. Based on the normalized area occupation, the parallel turbo decoder with \( P = 64 \) in this work have area overheads of 19.4% and 25.2% compared to the works from [12] with \( P = 64 \) and [14] with \( P = 32 \) respectively. Similarly, the postlayout simulation of our design with \( P = 8 \), in 90 nm CMOS technology, have 57% better throughput and 65.6% area overhead in comparison with the on-chip measured results of [10]. On the other hand, the parallel turbo decoder with \( P = 64 \) of this work has 38.4% better throughput as compared to the work [7] which is postlayout simulated in 90 nm CMOS technology. In between the parallel turbo decoders with \( P = 8 \) presented in this work and on-chip measured results of [11], we have achieved 11.2% better throughput while decoding for 5.5 full-iterations. Parallel turbo decoders implemented in this work are energy efficient, since they have achieved energy efficiencies of 0.11 nJ/bit/iterations and 0.079 nJ/bit/iterations for eight full-iterations with the configuration \( P = 8 \) and \( P = 64 \) respectively.

VI. CONCLUSION

This paper highlights the concept of modified sliding window approach and state metric normalization technique which resulted in a highly pipelined architecture of parallel turbo decoder. These techniques have specifically shortened the critical path delay and improved the operating clock frequency that has eventually aided the parallel turbo decoder to achieve higher throughput. Power issue of this design was mitigated using fine grain clock gating technique during the implementation phase.

constructed using six metal layers and integrated with programmable digital input-output pads as well as bonded pads. It has a core area of 6.1 mm² with the utilization of 86.9% and a gate count of 694 k. Similarly, we have carried out the synthesis-study as well as postlayout simulation for parallel turbo decoder with \( P = 64 \) in 90 nm CMOS technology and the layout of this implemented decoder is shown in Fig. 11(b). As discussed earlier, the value of \( M = 96 \) has been chosen for this design and it has increased achievable throughput as well as area overhead. In order to maintain a clock frequency of 625 MHz with increased parallelism, the ICNW is more complex and it imposes pipelined delay of 19 clock cycles. Similarly, deep-pipelined decoding delay \( \delta_d_{c,r} \) has increased to 394 clock cycles using (13). Based on (1), this decoder with \( P = 64 \) can achieve throughputs of 3.3 Gbps and 2.3 Gbps for 5.5 and 8 full-iterations respectively. However, it requires a core-area of 19.75 mm² with 5304 k gate count and consumes total power of 1450.5 mW.

Table III summarizes the key characteristics of implemented decoders of our work and compares them with the reported parallel turbo-decoder implementations in the literature [7], [8], [10]–[15]. These contributions include on-chip measured and postlayout simulated results in 65 nm, 90 nm and 130 nm CMOS technologies. Normalized area occupation and energy efficiency are also included in Table III for fair comparison. Among the contributions in 65 nm CMOS technology, the postlayout simulation of parallel turbo decoder with \( P = 64 \) has 29% better throughput than the throughput reported in [14]. Based on the normalized area occupation, the parallel turbo decoder with \( P = 64 \) in this work have area overheads of 19.4% and 25.2% compared to the works from [12] with \( P = 64 \) and [14] with \( P = 32 \) respectively. Similarly, the postlayout simulation of our design with \( P = 8 \), in 90 nm CMOS technology, have 57% better throughput and 65.6% area overhead in comparison with the on-chip measured results of [10]. On the other hand, the parallel turbo decoder with \( P = 64 \) of this work has 38.4% better throughput as compared to the work [7] which is postlayout simulated in 90 nm CMOS technology. In between the parallel turbo decoders with \( P = 8 \) presented in this work and on-chip measured results of [11], we have achieved 11.2% better throughput while decoding for 5.5 full-iterations. Parallel turbo decoders implemented in this work are energy efficient, since they have achieved energy efficiencies of 0.11 nJ/bit/iterations and 0.079 nJ/bit/iterations for eight full-iterations with the configuration \( P = 8 \) and \( P = 64 \) respectively.

VI. CONCLUSION

This paper highlights the concept of modified sliding window approach and state metric normalization technique which resulted in a highly pipelined architecture of parallel turbo decoder. These techniques have specifically shortened the critical path delay and improved the operating clock frequency that has eventually aided the parallel turbo decoder to achieve higher throughput. Power issue of this design was mitigated using fine grain clock gating technique during the implementation phase.
Similarly, large design-area of the decoder can be taken care by scaling down the technology. At 90 nm CMOS technology, an implementation of $8 \times$ parallel turbo decoder with radix-2 MAP decoders has achieved a maximum throughput of 446 Mbps with 5.5 iterations. Subsequently, the synthesis and postlayout simulation of parallel turbo decoder with 64 $\times$ radix-2 MAP decoders have shown a throughput of 3.3 Gbps which is suitable for 3GPP-LTE-Advanced as per its specification.

ACKNOWLEDGMENT

This work was carried out using resources from special manpower development programme II project sponsored by the Department of Information Technology India at the Indian Institute of Technology Guwahati. The authors would like to thank all the reviewers for their valuable comments, which have immensely helped to carry out more work and rewrite the paper with a broader perspective.

REFERENCES


Rahul Shrestha (S’13) received the B.Eng. degree in telecommunication engineering from the B.M.S. College of Engineering, Bangalore, India. He joined the Indian Institute of Technology, Guwahati for a Ph.D program in 2009.

He has been pursuing his research work since then, which includes VLSI design and ASIC/FPGA implementation of high-speed digital architectures for wireless communication applications. His research interest also comprises the study of channel codes from algorithmic and implementation perspectives.

Roy P. Paily (M’05) received the B.Tech. degree in electronics and communication engineering from the College of Engineering, Trivandrum, India in 1990, and the M.Tech. and Ph.D degrees from the Indian Institute of Technology, Kanpur and Indian Institute of Technology, Madras in 1996 and 2004 respectively, in the area of semiconductor devices.

He is currently a Professor in the Department of Electronics and Electrical Engineering, and the Head of the Centre of Nanotechnology, Indian Institute of Technology Guwahati, Guwahati, India. His research interests are VLSI circuits, MEMS, and devices.