Reversed-Trellis Tail-Biting Convolutional Code (RT-TBCC) Decoder Architecture Design for LTE

Trio Adiono, Ahmad Zaky Ramdani, Rachmad Vidya Wicaksana Putra
University Center of Excellence on Microelectronics, Institut Teknologi Bandung, Indonesia

Article Info

Article history:

Received Mar 8, 2017
Revised Nov 3, 2017
Accepted Nov 17, 2017

Keyword:

LTE
Maximum likelihood
Reversed-trellis algorithm
Tail biting convolutional code
VLSI architecture

ABSTRACT

Tail-biting convolutional codes (TBCC) have been extensively applied in communication systems. This method is implemented by replacing the fixed-tail with tail-biting data. This concept is needed to achieve an effective decoding computation. Unfortunately, it makes the decoding computation becomes more complex. Hence, several algorithms have been developed to overcome this issue in which most of them are implemented iteratively with uncertain number of iteration. In this paper, we propose a VLSI architecture to implement our proposed reversed-trellis TBCC (RT-TBCC) algorithm. This algorithm is designed by modifying direct-terminating maximum-likelihood (ML) decoding process to achieve better correction rate. The purpose is to offer an alternative solution for tail-biting convolutional code decoding process with less number of computation compared to the existing solution. The proposed architecture has been evaluated for LTE standard and it significantly reduces the computational time and resources compared to the existing direct-terminating ML decoder. For evaluations on functionality and Bit Error Rate (BER) analysis, several simulations, System-on-Chip (SoC) implementation and synthesis in FPGA are performed.

Copyright © 2018 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Trio Adiono,
University Center of Excellence on Microelectronics,
Institut Teknologi Bandung,
Ganesha Street No. 10, Bandung 40132, Indonesia.
Email: tadiono@stei.itb.ac.id

1. INTRODUCTION

The Viterbi algorithm has a crucial part in a convolutional code (CC) decoding process [1]. It is highly respected and extensively used because of its high coding gain for convolutional code decoding [2]. Viterbi algorithm is an optimum algorithm that uses the selection of state and data to find the highest probability of reconstructed data through trellis diagram. Selection is based on total inequality value between each state line value (i.e. branch metric) and received codewords. Viterbi algorithm works by comparing the received codewords with prediction data for determining the valid reconstructed bit [3]. Sequence with the minimum difference to the received codewords is selected as maximum-likelihood (ML) data. It means that the corresponding sequence is considered as reconstructed data.

Convolutional code has three schemes related to its initial state, namely direct-terminating (unfixed-tail) CC, fixed-tail CC, and tail-biting CC (TBCC) [4]. The direct-terminating based encoding is unchained processes. Both of initial and final states are not chained in a circular scheme. It is given a freedom to contain any data without any correlation. This scheme offers high coding efficiency, although it may result start-up errors. The predecessor path may meet with the valid path and the rest of bits are eventually decoded correctly. It is contrarian to the fixed-tail based scheme. Here, the information data are framed by several additional padding bits before encoding. The additional bits are used as initial state and recognized by the decoder. The conventional bits used are all zero (i.e. zero padding). It is effective in improving survivor routes convergence, but inefficient data rate emerges as issue to be dealt with since the non-informative data is added as initial state [4]. Lastly, in the TBCC, the encoded data must be formed from identical initial and final state to assure that received code is the valid one [4,5]. This scheme is considered to cover the advantages of both, direct-terminating (unfixed-tail) and fixed-tail CC because it uses informative data in initial state that help to decode the information better.

In order to implement TBCC, there are various sub-optimum techniques based on the Viterbi algorithm such as Circular Viterbi Algorithm (CVA) [6], Warped Around Viterbi Algorithm (WAVA) [7], Bidirectional Viterbi Algorithm (BVA) [8], improvement of CVA [7] and the combination of Viterbi and A* Algorithm [9]. As an alternative, we also proposed a technique that suits the hardware implementation named Reversed-Trellis TBCC (RT-TBCC) algorithm [4]. This algorithm is non-iterative and designed for minimizing the number of computation, thus a more efficient hardware implementation can be obtained. It overcomes start-up errors on the direct-terminating CC by separating the processing trellis into two parts, start-up and common area. It computes the start-up and common area first and collect all of the required data. Afterwards, the reverse decoding is performed until reaching the initial state. The data with minimum error is selected as valid data.

Arguably, the non-iterative algorithm is important for hardware implementation because it has a minimum and predictable number of computation. Thus, the power consumption is efficient and predictable as well. The iterative one could make the number of computation unpredictable. For example, one data frame could be decoded for less than *x* iteration, but there is no guarantee that the upcoming data frame could be decoded at the same number of iteration since it depends on the result. As long as the result is not tail-biting, the process keeps running. This may cause higher power consumption to decode the same size of data frame. The proposed algorithm tries to deal with that problem by minimizing the number of computation while maintaining the accuracy of tail-biting process. Hopefully, it is a decent alternative algorithm to perform an efficient tail-biting decoding process.

In this research, we aim to implement the proposed RT-TBCC algorithm in a VLSI architecture. It is obtained by using three steps on design methodology, namely processing element (PE) design, reversed-trellis unit (RTU) design and top-level integration. For real-time LTE standard application, plug-and-play approach is used for integrating the RT-TBCC VLSI module into LEON3-based System-on-Chip (SoC).

This paper is organized in a number of sections. First is introduction of the research background. Second is related theories. It is followed by explanation on the RT-TBCC algorithm in the third section. A step-by-step description of the algorithm is also presented. The fourth section is the proposed modular hardware and System-on-Chip (SoC) implementations for LTE application. Results and discussions are given in the fifth section. Lastly, conclusions, acknowledgment, and references are given in the three two sections, respectively.

2. RELATED THEORIES

Convolutional encoding process is usually characterized in (***c***, ***n***, ***m***), with ***n***/***c*** is code rate (***R***) and ***m*** is constraint length. There are three kinds of CC based on its initial state, namely direct-terminating (unfixed-tail) CC, fixed-tail CC, and tail-biting convolutional codes (TBCC). Each of them has a unique approach of computation regarding its target of design.

The direct-terminating CC is unchained encoding process. Thus, if it is declared as (***c*****,** 1**,** ***m***), then for each sequence of length ***L***, it generates a codeword with length ***c*** **×** ***L*** because no additional data in the encoding process. In the fixed-tail CC scheme, information data are framed by several additional padding bits before encoding process. For data length ***L***, it produces a codeword (***L***+***m***) **×** ***c*** so that there are losses of data rate by ***m*** / (***L***+***m***). Lastly, the TBCC generates codeword of length (***L***+***m***) × ***c***, similar with fixed-tail CC, but the data length is (***L***+***m***) instead of ***L*** because the initial state is not additional bits but the end-part of the data itself. A complete illustration of these three schemes can be seen in Figure 1.

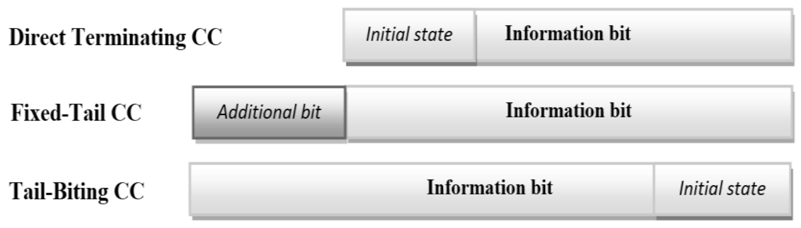


Figure 1. Differences between three initial-state scenarios in CC process

3. REVERSED-TRELLIS TBCC ALGORITHM

The idea of reversed-trellis TBCC (RT-TBCC) algorithm is to find how tail-biting based Viterbi decoding process can be finished without iteration or at least has a fixed processing time [10]. Since the RT-TBCC algorithm is established from TBCC concept, it also utilizes the same initial and end states concept. The difference is that the proposed RT-TBCC exploits the advantages in fixed-tail and direct-terminating concepts to determine the minimum error trellis path. From three schemes explained before (Figure 1), we find that the differences between the direct-terminating and fixed-tail based decoding only occur on the trellis path $\boldsymbol{k} < \boldsymbol{m}$, with $\boldsymbol{k}$ is trellis order and $\boldsymbol{m}$ is constraint length. Meanwhile, in area $\boldsymbol{k} > \boldsymbol{m}$, the trellis processes are common [4]. These are illustrated in Figure 2 and Figure 3. In other words, we can simply state that RT-TBCC algorithm performing two processes of Viterbi algorithm on two opposite directions. It is similar with BVA algorithm [8]. The main difference is that the BVA performs bidirectional Viterbi algorithm from the middle of trellis, meanwhile the RT-TBCC performs bidirectional Viterbi algorithm from $\boldsymbol{m}$ instead of the middle of trellis. In the RT-TBCC, there are four stages involved based on the operational separation [4].

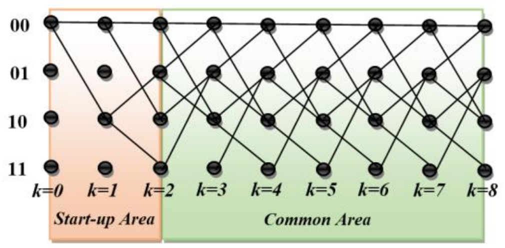


Figure 2. Fixed-tail CC trellis with ‘00’

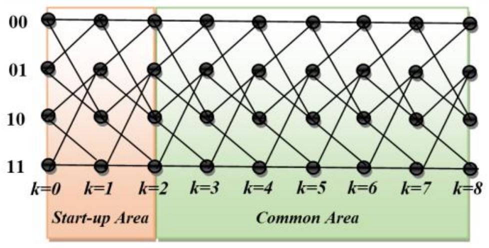


Figure 3. Direct-terminating CC trellis

The first process is to execute a direct-terminating Viterbi algorithm with an assumption that all states could be the valid initial state. This process is executed for $0 < \boldsymbol{k} < \boldsymbol{m}$ (i.e. start-up stage). Illustration is given in Figure 4 for case-study $\boldsymbol{m} = 2$. In this process, the received data are stored for next stage computation. If there is path metric ($\boldsymbol{PM_{t,k}}$) value for state $\boldsymbol{t}$ at time $\boldsymbol{k}$, and a branch metric ($\boldsymbol{BM_{t',k}}$) value from state transition $\boldsymbol{t'} \rightarrow \boldsymbol{t}$, with $\boldsymbol{t'}$ and $\boldsymbol{t''}$ are previous state from $\boldsymbol{t}$, therefore (1) is applied. From (1), we can calculate the cost of each path in this first step as (2).

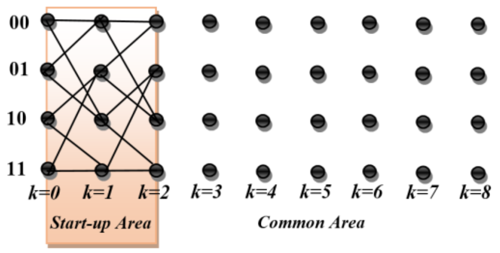


Figure 4. RT-TBCC first stage – start-up

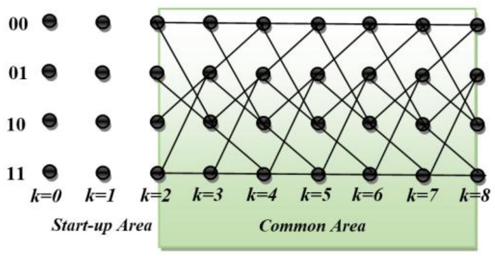


Figure 5. RT-TBCC second stage – common

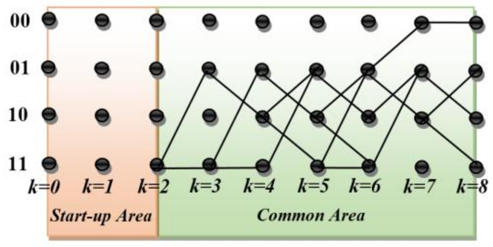


Figure 6. Convergent starting state at “11” and $\boldsymbol{k} = \boldsymbol{m} = 2$

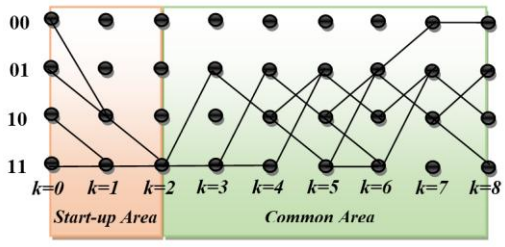


Figure 7. All possible tail-biting routes

In the second stage (i.e. common stage), the similar processes are conducted. Results of first stage calculations are not involved here, except only for the calculated cost. Because the processes are similar to the first stage, the cost calculation is also similar (3). The illustration of second stage is given in Figure 5. It shows that the processes are working in area ***m*** < ***k*** < ***L***. They are expected to result survivor routes for every last state. If errors do not affect significantly, the survivor routes are convergent into a single starting state (at ***k*** = ***m***) as illustrated in Figure 6. Otherwise, if errors affect significantly, the survivor routes are divergent into several starting states.

$$PM_{t,(k+1)} = \min\left\{\left(PM_{t',(k)} + BM_{t',(k)}\right), \left(PM_{t'',(k)} + BM_{t'',(k)}\right)\right\} \tag{1}$$

$$R_0 = W_{t,(0,m)} = \sum_{k=0}^{m} PM_{t,k} \tag{2}$$

$$R_t = W_{t,(0,L)} = \sum_{k=0}^{L} PM_{t,k} \tag{3}$$

In the third stage, tail-biting process is performed. Here, we force all of the surviving routes that end in each state to do reverse-trellis process by tracking the path to corresponding initial states so that the particular trellis ends in the initial state that has the same value of its end state. It is illustrated in Figure 7. From the example, there are four possible routes to consider as valid route: tail-biting at states "00", "01", "10", and "11". For the last stage, cost calculation is performed for each tail-biting scenario. For each path at 0 < ***k*** < ***m***, the cost is calculated based on the stored data in the first stage (i.e. start-up stage). Final cost is calculated using (4), the common route cost $\boldsymbol{R_t}$ is subtracted by start-up route cost $\boldsymbol{R_0}$ and added by tail biting cost $\boldsymbol{R_{tb}}$. The subtraction is performed because previous ***k*** < ***m*** path produced from Viterbi algorithm (first stage) is replaced with new ***k*** < ***m*** path to produce a tail-biting route (third stage). This total cost is the primary parameter for determining the ML path for final selection.

The calculations performed in a reversed-trellis process are actually similar to the process in add-compare-select (ACS). If the ACS aims to choose one of the predecessor states with minimum cost, the reversed-trellis does not need to choose the states because the order of predecessor states have been defined before at start-up stage. In order to help understanding the process, illustration is provided in Figure 8.

$$R_{total-t} = R_t - R_o + R_{tbt} \tag{4}$$

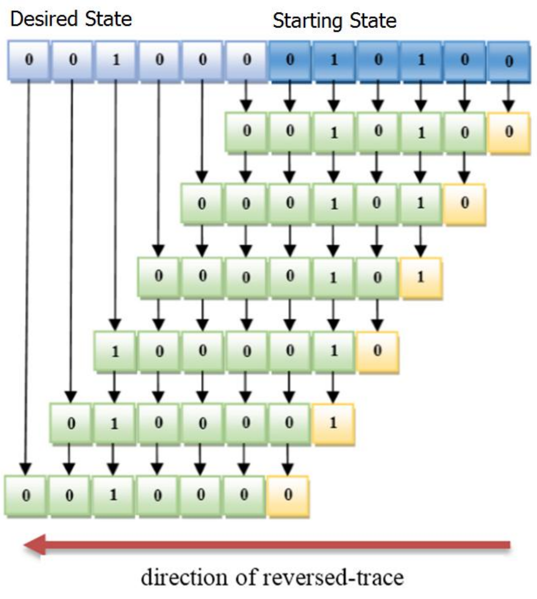


Figure 8. Illustration of reversed-trellis state process

4. PROPOSED HARDWARE ARCHITECTURE
4.1. LTE Target Application

Application target is Long Term Evolution (LTE) communication standard. In LTE, TBCC is used in two downlink channels, namely the Physical Broadcast Channel (PBCH) and the Physical Downlink Control Channel (PDCCH). Each frame consists of 40 bits data. The TBCC encoder used in the LTE standard is shown in Figure 9. Its encoding rate ***R*** is equal to 1⁄3. It means that the 3 bits results are produced for each bit of input. Constraint length is 7 and equal to number of shift-register (i.e. 6) + 1. After encoding process, codewords are obtained and then they are transmitted in OFDM-QPSK modulation.

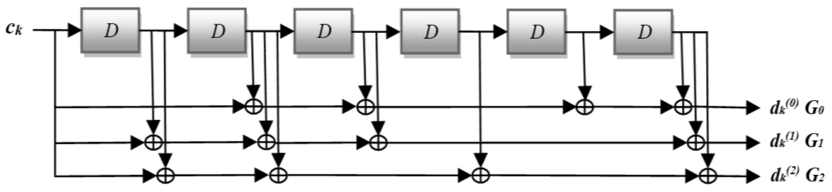


Figure 9. LTE TBCC encoder

4.2. Processing Element

Processing element (PE) is an element that compute the input data in order to obtain the best path that come to a branch of state. PE comprises 2 elements, namely branch metric unit (BMU) and add-compare-select (ACS). The internal architecture of PE is presented in Figure 10. Each PE consists of 1 BMU and 8 ACS because of inter-states butterfly-like mechanism. BMU is responsible to deliver combination of valid data as input of every state. Each state's output possibilities are computed to obtain hamming distance in ACS unit. ACS calculates the costs of 2 paths which come to the same state and these costs are added to the costs of current output possibilities. The data with minimum cost is selected as reference of survivor path. Each ACS structure comprises 2 hamming calculators, 2 adders, a comparator and a multiplexer. Those two hamming calculators and adders aim to conduct a parallel computation for hamming distances and costs. On the other hand, the comparator selects the survivor. Detail of ACS internal architecture is presented in Figure 11. Output of the ACS are minimum routing cost and minimum path flag.

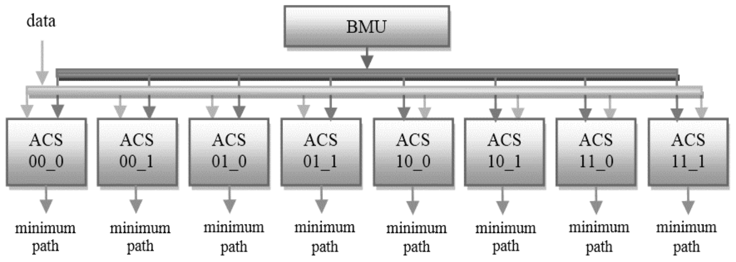


Figure 10. PE internal architecture

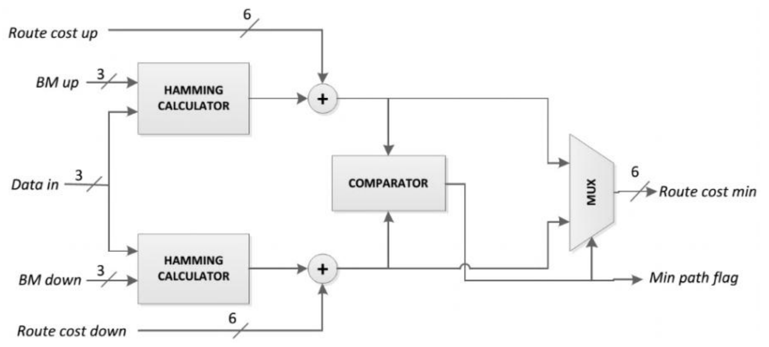


Figure 11. ACS internal structure

4.3. Path Metric Unit

Path Metric Unit (PMU) is responsible for storing the route matrices which have been passed through. Records of routing paths and initial states are stored here. Thus, we can easily recall them when needed. PMU is established from modular Path Metric Cell (PMC) unit. This unit receives two input ports for state-butterfly pair function and a flag signal. This flag is produced by ACS and passed to PMC for selecting a prospective path for each state. Internal structure of PMC is shown in Figure 12. Each PMC unit is connected to each other to mimic butterfly-like pair structure for trellis computation. Number of PMCs depend on the state used for the computation.

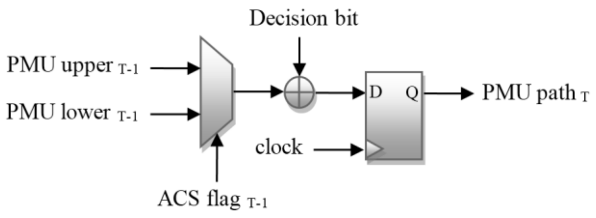


Figure 12. PMC internal structure

4.4. Best State Unit

Best State Unit (BSU) is responsible to compute the total costs and result minimum cost with its corresponding routing path. It receives total costs information after all processing are finished. There are two kind of costs which come from common stage and reversed-trellis stage processes. In order to find the minimum cost, sorting is used. Sorting processes are done for 64 paths from each state. Sorting process is conducted if the traceback processes are finished. Since the proposed architecture is designed for LTE standard, the total number of traceback steps are 40.

4.5. Reversed-Trellis Unit

Reverse trellis unit (RTU) work scheme is to calculate the path cost in area $\boldsymbol{k} < \boldsymbol{m}$, which the paths have been determined earlier and stored during the process of Viterbi algorithm on the first stage. Illustration of the complete reversed-trellis block diagram can be seen in Figure 13. There are five modules inside RTU, namely first sequence buffer, branch metric calculator, route shift register, hamming calculator and tail-biting cost. Data input are stored in the first sequence buffer until RTU ready to process new data block. If the Viterbi decoding process is finished, information data from PMU are used as routing reference for cost calculation. The bit-shift routing and its cost calculation, as illustrated in Figure 8, are done in route shift register and branch metric calculator. After each bit-shifting process, cost of the route is included to hamming calculator and combined with the cost generated from stage-two, in order to get the final value. The final value is calculated in tail-biting cost module and the minimum routing cost is considered as the valid reconstructed data.

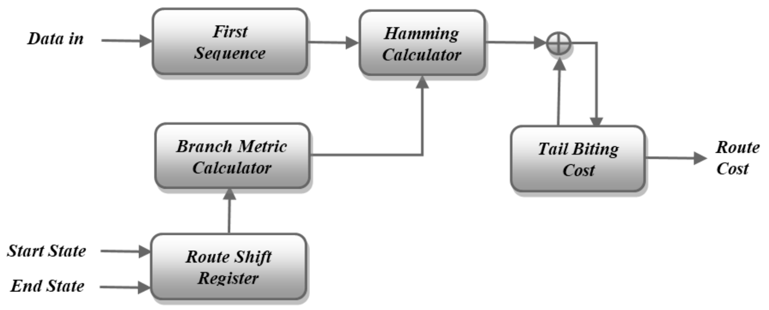


Figure 13. Reversed-trellis block diagram

4.6. Top-Level Architecture

Since the application target is LTE standard, we need 100 Mbps throughput in the system [11], [12].

In order to achieve this target, system architecture with 64 parallel ACSs are needed. Design of RT-TBCC decoder comprises seven components, namely input register, PE, PMU, RTU, BSU, output register and main control. They are organized as illustrated in Figure 14. Input data go to input register and buffers. Each data has 3 bits width, since the CC has 1/3 rate of convolution. These data are received by PE and RTU. RTU saves data $\boldsymbol{d_0} - \boldsymbol{d_m}$ in every process for reverse trellis process. Since we use 40 bits data original, we need to provide 120 bits for processing. The data received by PE are processed in ACS and BMU modules. Each PE consists of 1 BMU and 8 ACS and it computes data paths and the corresponding costs. The minimum cost and its corresponding path is passed to PMU and BSU. Information passed to PMU is neither cost value nor trellis path record, but just a flag signal. Meanwhile, BSU receives costs for existing paths. If the data are processed successfully, RTU then calculates the cost of path-forming and cost of existing paths from BSU. The state with minimum cost is selected as the last survivor or valid data. Detailed timing diagram of these processes is presented in Figure 15.

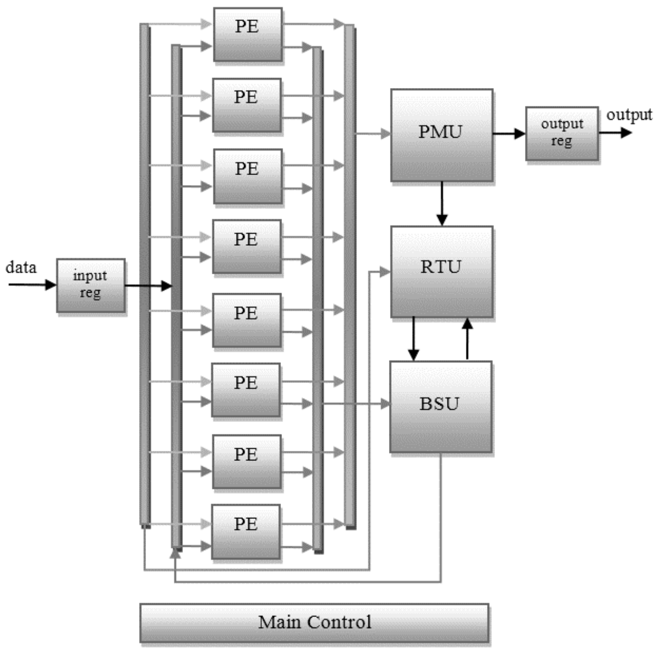


Figure 14. Top-level of RT-TBCC decoder

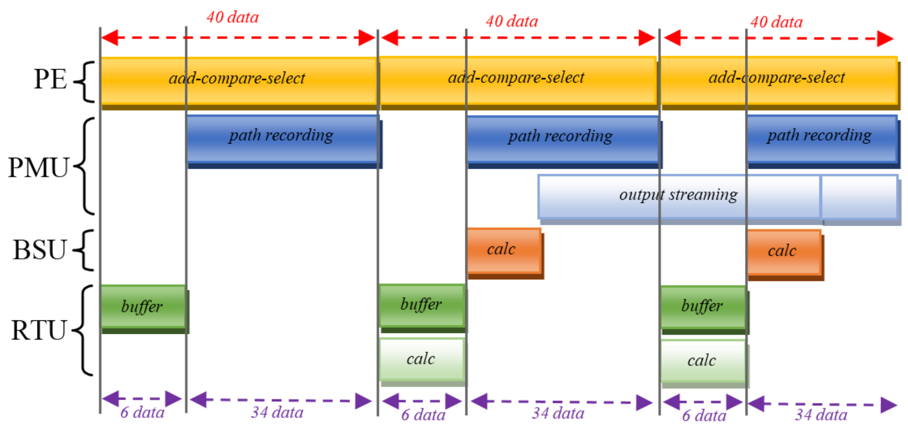


Figure 15. Timing diagram of the RT-TBCC

4.7. System-on-Chip Implementation

System on Chip (SoC) technology can support resources management which is crucial for real-time applications. This feature makes SoC has successfully driven many electronic based product developments [13]. Various devices for diverse applications are basically established from SoC technology [14]. To utilize the error correction capability in real-time applications, CC decoder is considered to be integrated in such a hardware-software co-design system [2]. Thus, in the proposed system, the RT-TBCC design is integrated into SoC as one of its modules. The proposed SoC design for RT-TBCC implementation is presented in Figure 16. We use LEON3 based SoC as reference design. We attach the RT-TBCC decoder as co-processor which is located as a slave module in AHB bus of SoC. Moreover, we also incorporate direct memory access (DMA) module for accelerating data transfer from RT-TBCC decoder, because we need to achieve 100 Mbps target speed for LTE standard.

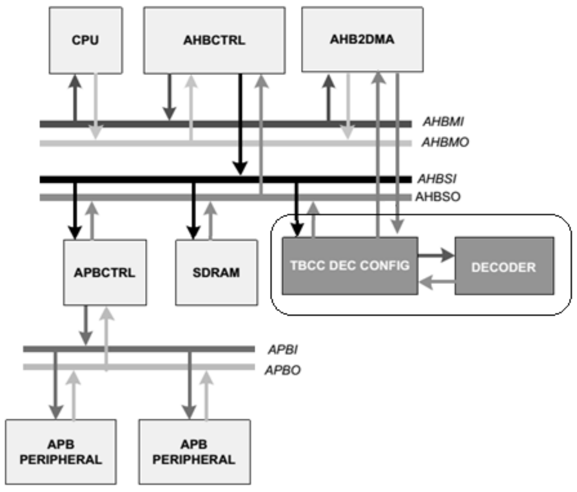


Figure 16. RT-TBCC SoC implementation

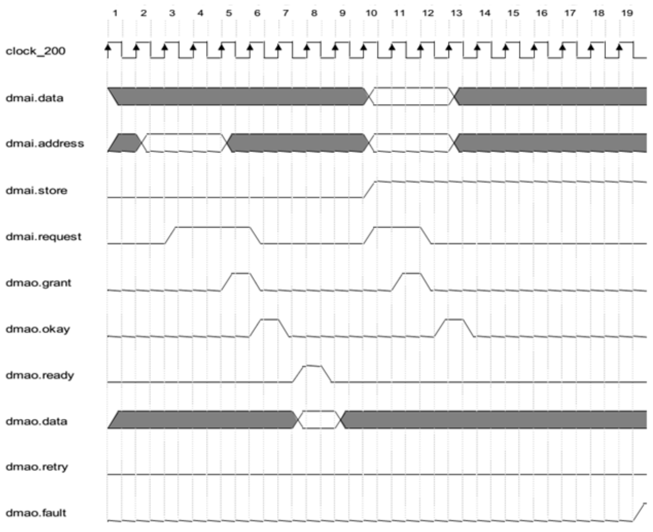


Figure 17. DMA access timing diagram

For DMA processing, it is started with providing valid data on the signals except *dmai.request* signal. After those the signals are valid, *dmai.request* is activated “high”. It means that the system is requesting DMA access. If *dmao.grant* signal responds with “high”, it means that the request has been granted. Afterwards, the system waits a response from *dmao.okay*, *dmao.retry* or *dmao.fault* signals. If there is no response yet, the *dmai.request* continues as “high”. If the one which respond “high” is *dmao.okay*, then the system continues to wait the *dmao.ready* signal. The *dmao.ready* valid signal means that the corresponding *dmao.data* signals are valid. Otherwise, if the one which respond “high” is *dmao.retry* or *dmao.fault*, then the request signal *dmai.request* needs to be sent again. Illustration of this process can be seen in Figure 17. Time period 2-8 is data reading process from DMA which is indicated with activated signals of *dmao.data* and followed by *dmao.ready*. Besides, time period 10-14 is data writing process into DMA. Furthermore, DMA module takes care the data transactions for decoder module as well.

5. RESULTS AND ANALYSIS

5.1. Bit Error Rate (BER) Analysis

The BER analysis is conducted in term of LTE application between proposed RT-TBCC algorithm vs traditional direct-terminating ML algorithm. The test flow is illustrated in Figure 18. Encoded and QPSK modulated data are transmitted in AWGN channel with variable value of signal-to-noise ratio (SNR). SNR level of the AWGN channel is the level of energy-per-symbol (*Eb/No*). Simulations are carried out to see BER of decoding result data for each SNR. Results of a simulation can be seen in Figure 19. Results show that the RT-TBCC algorithm can improve the ML correction ability. This improvement is shown by better BER performance. At the time when SNR value is low, improvement on BER is not significant. But, the improvement on BER performance is growing along with the increasing of SNR.

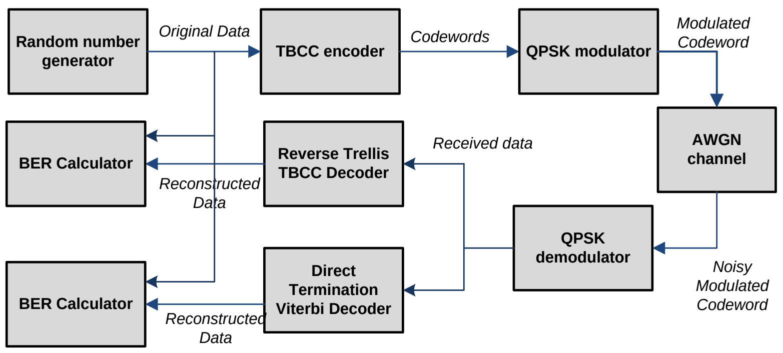


Figure 18. Test flow between RT-TBCC vs direct-terminating ML

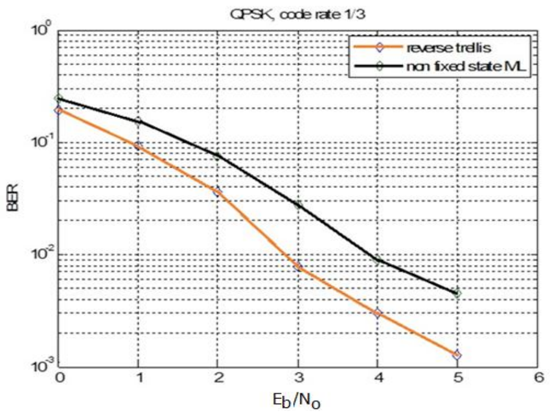


Figure 19. BER performance between RT-TBCC vs direct-terminating ML

5.2. Computational Reduction

We implement the proposed algorithm for LTE standard, thus we use data block length of 40 bits. Comparisons are made between the RT-TBCC with brute force and fixed-tail ML algorithms. For a decoding process, RT-TBCC consists of direct-termination process of Viterbi algorithm and continues with tail-biting routes. It needs 64 states that perform 40 times of ACS process comprising addition and comparison. We can simply see an ACS process as two additions and one inversion. Thus, the total operation required at the first stage in RT-TBCC is 64 × 40 × 2 = 5,120 additions and 64 × 40 × 1 = 2,560 inversion. In the second stage, computation process is to calculate tail-biting forming costs. Since it needs $\boldsymbol{m} = 6$, the second stage requires $64 \times 6 = 384$ addition. In the total cost calculation of each survivor, 64 states require $64 \times 2 = 144$ additions and 64 inversions. Furthermore, the best path is achieved by comparing 64 survivor routes. Thus, the total computations in the RT-TBCC decoding process is 5,712 additions and 3,008 inversions.

For brute force, calculations are performed for all possible paths. Each state has as many as the number of possible routes $2^{(\boldsymbol{L}-\boldsymbol{m})} = 2^{40-6} = 17{,}179{,}869{,}184$. Hence, the total possible routes for all possible initial state are 1,099,511,627,776. For the calculation, each route consists of 40 processes. In order to get the best route comparison, it needs as many as the number of routes, so the total computation required to perform the decoding process is 45,079,976,738,816 additions and 1,099,511,627,776 inversions. It is obviously impractical to do. Lastly, a fixed-tail ML decoder without tail-biting consideration, consists of 64 iterations (possible initial states). Each iteration process occurs for each ACS states on $6 < \boldsymbol{k} < 40$, because when $\boldsymbol{k} < 6$, the available states are limited. Thus, for each iteration required, number of ACS operation is $(64 \times 34) + (1 + 2 + 4 + 8 + 16 + 32) = 2{,}239$. It equals to 4478 additions and 2,239 inversions. The total for the entire iterations take 286,592 additions and 143,296 inversions. If we calculate it together with the best route searching operation, it requires a total of 286,736 additions and 143,360 inversions.

5.3. SoC Evaluation

SoC evaluation is done by performing simulations. The results are shown in Figure 20 and Figure 21. This evaluation involves 200 data frames with random errors. The summary of evaluations is presented in Table 1. Meanwhile, architectural hardware evaluations are presented in Table 2. We can see in Figure 20 that the simulation result gives the same value compared to the signal tap result from the FPGA implementation. In Figure 21, we also see that the system can achieve error 0x00 from LEON3 SoC monitoring software. These mean that the original data can be perfectly reconstructed. But, for large number of errors, the system could not perfectly reconstruct the data. It may occur because of the limitation of coding effectivity. But, from architectural aspect, the proposed SoC design can perform as expected, because it reaches 100 Mbps throughput performance for LTE communication standard. It is proven by architectural evaluations in Table 2. For FPGA implementation in both Altera DE2 and DE4, the proposed design can achieve > 100 MHz frequency. Since it delivers output data each clock cycle, the LTE communication standard with 100 Mbps data throughput can be provided by the proposed design.

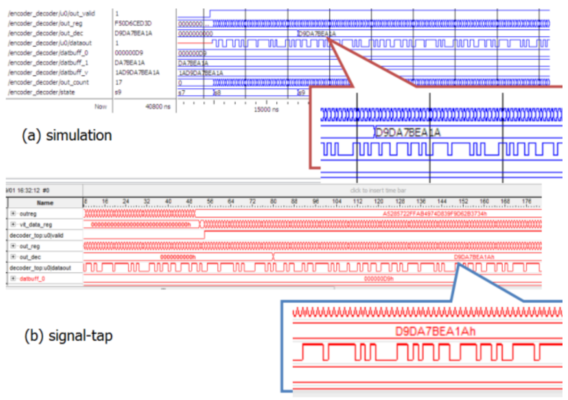
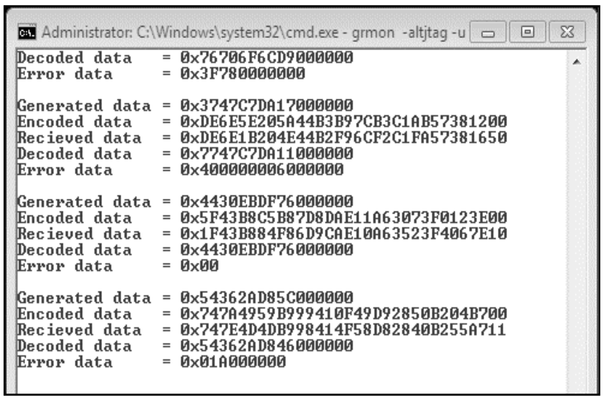


Figure 20. Results from simulation vs signal-tap



```
Administrator: C:\Windows\system32\cmd.exe - grmon  -altjtag -u
Decoded data    = 0x76706F6CD9000000
Error data      = 0x3F780000000

Generated data = 0x3747C7DA17000000
Encoded data    = 0xDE6E5E205A44B3B97CB3C1AB57381200
Recieved data   = 0xDE6E1B204E44B2F96CF2C1FA57381650
Decoded data    = 0x7747C7DA11000000
Error data      = 0x4000000006000000

Generated data = 0x4430EBDF76000000
Encoded data    = 0x5F43B8C5B87D8DAE11A63073F0123E00
Recieved data   = 0x1F43B884F86D9CAE10A63523F4067E10
Decoded data    = 0x4430EBDF76000000
Error data      = 0x00

Generated data = 0x54362AD85C000000
Encoded data    = 0x747A4959B999410F49D92850B204B700
Recieved data   = 0x747E4D4DB998414F58D82840B255A711
Decoded data    = 0x54362AD846000000
Error data      = 0x01A000000
```

Figure 21. Simulation results monitor

Table 1. Data evaluation summary.

Parameters	Information Sent	After Decoding Process
Number of Data Frame	200	200
Error Bit / Total Bit	11913 / 24000	361 / 8000
BER	0.496	0.045

Table 2. Architectural hardware summary.

FPGA	Area		Max Frequency	Latency
Altera Board	LEs	Registers	(MHz)	(clock cycles)
DE2	8,813	7,274	123.4	60
DE4	11,325	7,274	212.0	

6. CONCLUSION

In this paper, we propose a VLSI architecture to implement our novel reversed-trellis TBCC (RT-TBCC) algorithm. It is a non-iterative algorithm developed from direct-terminating ML process with tail-biting concept. It aims to minimize the number of computation for direct-terminating maximum-likelihood (ML) decoding process. The RT-TBCC algorithm enables a prediction on the number of computation needed in the process. The proposed architecture for LTE communication standard demonstrates promising results in hardware implementation and performance. It requires only 5,712 addition and 3,008 inversion. It is a significant decrease compared to 286,592 addition and 143,296 inversion for fixed-tail ML decoder and 45,079,976,738,816 addition and 1,099,511,627,776 inversion for brute force algorithm. Moreover, its SoC implementation can perform computation with 100 Mbps throughput as expected for LTE.

ACKNOWLEGEMENTS

This article is extended version of papers entitled “A novel algorithm of tail biting convolutional code decoder for low cost hardware implementation” and “Tail biting convolutional code decoder co-processor for high throughput System-on-Chip” that were published at ISPACS 2015 [4] and ISOCC 2015 [10] respectively.

REFERENCES

[1] A. J. Viterbi, “Error Bounds for Convolutional Coding and an Asymptotically Optimum Decoding Algorithm”, *IEEE Transactions on Information Theory (TIT)*, vol. 13, no. 2, pp. 260-269, April 1967.

[2] R. V. W. Putra and T. Adiono, “VLSI Architecture for Configurable and Low-Complexity Design of Hard-Decision Viterbi Decoding Algorithm”, *Journal of ICT Research and Applications (JICTRA)*, vol. 10, no. 1, pp. 57-75, March 2016.

[3] R. V. W. Putra and T. Adiono, *“A Configurable and Low Complexity Hard-Decision Viterbi Decoder in VLSI Architecture”*, Proc. of 2014 2nd International Conference on Information and Communication Technology (ICoICT), pp. 177-181, May 2014.

[4] A. Z. Ramdani and T. Adiono, *“A Novel Algorithm of Tail Biting Convolutional Code Decoder for Low Cost Hardware Implementation”*, Proc. of 2015 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp. 241-245, November 2015.

[5] H. H. Ma and J. K. Wolf, “On Tail Biting Convolutional Codes”, *IEEE Transactions on Communications (TCOM)*, vol. 34, no. 2, pp. 104-111, February 1986.

[6] R. V. Cox and C. E. Sundberg, “An Efficient Adaptive Circular Viterbi Algorithm for Decoding Generalized Tailbiting Convolutional Codes”, *IEEE Transactions on Vehicular Technology*, vol. 43, pp. 57-68, February 1994.

[7] R. Shao, S. Lin, and M. Fossorier, “Two Decoding Algorithms for Tailbiting Codes”, *IEEE Transactions on Commununications (TCOMM)*, vol. 51, no. 10, pp. 1658-1665, October 2003.

[8] X. Wang, H. Qian, J. Xu, Y. Yang, and F. Wang, *“An Efficient CVA-based Decoding Algorithm for Tail-Biting Codes”,* Proc. of IEEE Global Telecommunications Conference (GLOBECOM), pp. 1-5, December 2011.

[9] P. Shankar, P. N. A. Kumar, K. Sasidharan, B. S. Rajan, and A. S. Madhu, “Efficient Convergent Maximum Likelihood Decoding on Tailbiting Trellises”, *CoRR*, vol. abs/cs/0601023, 2006. URL: http://arxiv.org/abs/cs/0601023v2

[10] A. Z. Ramdani and T. Adiono, *”Tail Biting Convolutional Code Decoder Co-Processor for High Throughput System-on-Chip,”* Proc. of 2015 International SoC Design Conference (ISOCC), pp. 303-304, November 2015.

[11] A. Z. Yonis and M. F. L. Abdullah, ”Uplink and Downlink of LTE-Release 10 in Celular Communications,” *Int. J. of Informatics and Communication Technology (IJ-ICT)*, vol. 1, no. 1, pp. 43-53, July 2012.

[12] A. Z. Yonis and M. F. L. Abdullah, ”Peak-Throughput of LTE-Release 10 for Up/Down Link Physical Layer,” *Int. J. of Information and Network Security (IJINS)*, vol. 1, no. 2, pp. 88-96, June 2012.

[13] R. V. W. Putra and T. Adiono, *“Hybrid Multi–System-on-Chip Architecture: A Rapid Development Design for High-Flexibility System”*, Proc. of 2016 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1-4, January 2016.

[14] R. V. W. Putra and T. Adiono, “Hybrid Multi–System-on-Chip Architecture as Rapid Development Approach for High-Flexibility System”, *IEIE Transactions on Smart Processing and Computing (IEIE-SPC)*, vol. 5, no. 1, pp. 55-62, February 2016.

BIOGRAPHIES OF AUTHORS

Trio Adiono received B.Eng. on Electrical Engineering and M.Eng. on Microelectronics from Bandung Institute of Technology (ITB), Indonesia, in 1994 and 1996. He obtained his Ph.D. degree in VLSI Design from Tokyo Institute of Technology, Japan, in 2002. He received the "Second Japan Intellectual Property Award" in 2000 from Nikkei BP for his research on "Low Bit-rate Video Communication LSI Design". He also holds a Japanese Patent on "High Quality Video Compression System". Currently, he is a lecturer at the School of Electrical Engineering and Informatics, a Head of the Microelectronics Center and IC Design Laboratory, ITB. He currently serves as a chair of the IEEE SSCS Indonesia Chapter. His research interests include VLSI, signal and image processing, smart card, electronics solution design and integration.

Ahmad Zaky Ramdani received B.Sc. on Telecommunication Engineering in 2011 from Telkom University, Indonesia, and M.Sc. on Microelectronics in 2015, from Bandung Institute of Technology (ITB), Indonesia. He was an engineer on embedded system and system control at Versatile Silicon Technology. He also experienced as a part-time researcher at Microelectronics Center ITB from 2014 to 2015. Currently, he is working as firmware engineer at Kulim Smart Technologies, Malaysia. His interests are mainly on VLSI, SoC, signal processing, embedded system and robust architecture.

Rachmad Vidya Wicaksana Putra received B.Sc. on Electrical Engineering in 2012 and M.Sc. on Microelectronics with distinction honor in 2015, both from Bandung Institute of Technology (ITB), Indonesia. He received an Indonesia Endowment Fund for Education (IEFE/LPDP) Scholarship for his master and doctoral study. He has been involved as teaching assistant at Electrical Engineering ITB (2010-2017), research assistant at Microelectronics Center ITB (2012-2017) and engineer at Fusi Global Technology (2012-2014). Currently, he is pursuing his doctoral research in computer architecture. His interests include VLSI, SoC, computer architecture, IoT and VLC.