Middle-East Journal of Scientific Research 23 (Sensing, Signal Processing and Security): 27-35, 2015
ISSN 1990-9233

DOI: 10.5829/idosi.mejsr.2015.23.ssps.10

Low-Complexity and Power Efficient VLSI Architecture for Turbo Decoder

R. Kaviyarasi and M. Selvi

Saveetha Engineering College, Chennai, India

Abstract: Turbo codes consumes less power consumption and are powerful error correcting code, hence utilized in power constrained wireless communication applications. In this work, implementation of turbo decoder is considered to reduce the area, delay and power using LUT (Look up Table) - Log - BCJR (Bahl-Cocke - Jelinek - Raviv) decoding algorithm. Add Compare select (ACS) unit, memory unit and twin level registers forms the basic architecture for LUT-Log-BCJR turbo decoder. This architecture is divided into essential add compare select operations employing a novel low-complexity ACS unit which uses RNS (Residue Number System) adder. This proposed system will reduce the number of gates needed and additionally facilitates a reduction in the power consumption and delay.

Key words: Turbo codes · Log-BCJR algorithm · Error Correcting Code (ECC) · Residue Number System (RNS) · Turbo code

INTRODUCTION

Turbo decoders are extensively utilized in wireless communication applications, exceptionally in wireless sensor networks that are projected for power constrained scenarios. High transmission throughput of about 100 Mbit/s is achieved in ASIC- based turbo decoder architecture [1, 2] rather than power using Max-Log-BCJR turbo decoding algorithm [3]. Data and voice calls simultaneously supported by 3GPP mobile wireless standard in unified turbo/Viterbi decoder architecture [4] uses the log MAP algorithm which yields higher bit error rate (BER) performance. The area facilitated using this algorithm is 9 mm^2 and power consumption is 292 mW which are yet to be reduced. High-speed 3G mobile data terminals uses a radix-4 log MAP turbo decoder [5] which have reduced complexity and fast operation with only 0.04dB turbo decoding loss. The area facilitated using this algorithm is 14.5 mm^2 and power consumption is 956 mW which are comparatively high.

Forward-error-correction (FEC) standards are satisfied by a unified Convolutional/Turbo decoder [6] where the Convolutional code and Turbo code co-exit. In VA/MAP (Viterbi algorithm/ Maximum a posterior) decoding functions, the timing analysis shows that processing element (PE) have 100% utilization rate. The area facilitated using this algorithm is 8.2 mm^2 and power consumption is 320 mW which are less than the previous work of paper [5]. Higher throughput is achieved by wireless communication standards enhanced by 3GPP long term evolution (LTE) [1]. These effectual decoder architectures for exceedingly punctured LTE Turbo codes achieve a throughput of about 150 Mbit/s. The area enabled employing this algorithm is 2.1 mm^2 and power consumption is 300 mW. Implementation methods of the parallel turbo-decoders [7] of bit rate 326.4 Mb/s operating parallel constitutes the multiple soft-input and output decoders. Radix-4-based 8 parallel turbo-decoder ASIC technologies yield a throughput of 390 Mb/s. The area facilitated by this algorithm is 3.57 mm^2 and power consumption is 332.8 mW. LUT-Log-BCJR architecture [8] is decomposed into add compare select (ACS) operations and ACS unit is used to perform those operations. The area facilitated using this algorithm is 3 mm^2 and power consumption is 89.08 mW which are further to be reduced.

Section II explains the Conventional architecture for LUT-Log-BCJR turbo decoder which cannot significantly reduce the area, power and delay. This motivates the novel architecture of section III, which specifically reduces the complexity, power and delay of the turbo decoder. Section IV shows the simulation and synthesis result of proposed architecture. Finally section V concludes the paper.

Corresponding Author: R. Kaviyarasi, Saveetha Engineering College, Chennai, India.

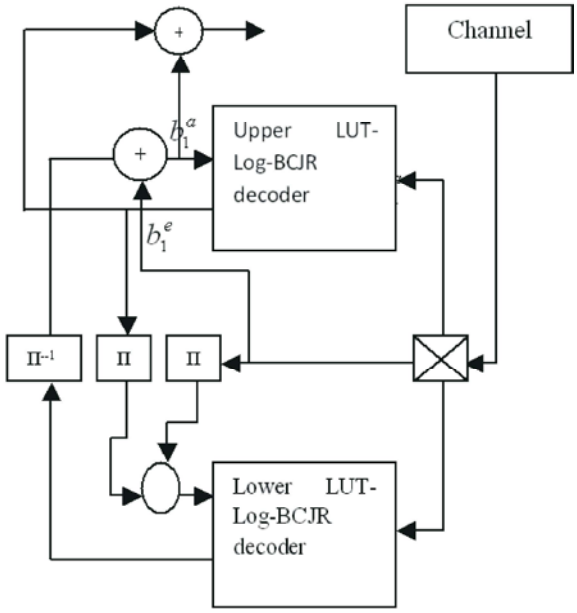


Fig. 1: Turbo Decoder Architecture

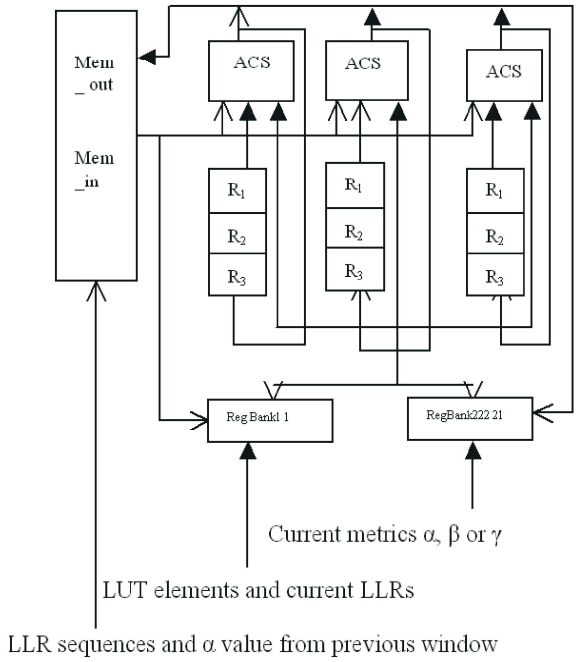


Fig. 2: LUT-Log-BCJR architecture

Conventional LUT-Log-BCJR Algorithm: Turbo decoder comprises of parallel concatenation of two convolutional decoder as shown in Fig. 1 which employ LUT-log-BCJR algorithm called LUT - log - BCJR decoder [9, 10]. This decoder processes the input as Logarithmic Likelihood Ratios (LLRs) [11] rather than operating on bits.

$$LLRb = \ln \frac{(P(b=0))}{(P(b=1)} \tag{1}$$

The input to this decoder is two a priori LLR sequences b_1^a and b_2^a which produces the output as extrinsic LLR sequence b_1^e which is given by

$$b_{1,j}^e = \max^*[\alpha_{j+1} + \sum_{i=2} \gamma_{i,j} + \beta_{j-1}] \tag{2}$$

The other LUT-Log-BCJR decoder uses this extrinsic LLR sequence as the a priori LLR sequence in the next iteration [12]. This cycle is repeated and with every iteration, the error correction performance is increased. Fig. 2 shows the existing LUT-Log-BCJR architecture that employs a sliding window technique [12, 13]. To calculate extrinsic LLR sequence b_1^e, LUT-log-BCJR algorithm is used which follows four steps.

- Compute Branch Metrics
- Compute Alpha Coefficient
- Compute Beta Coefficient
- Compute LLR as Output
- Compute Branch Metrics

The branch metrics [14, (2)] to the present window are generated which is given by the formula,

$$\gamma_{i,j} = \ln P(y_j / x_j) + \ln P(b_{i,j}) \tag{3}$$

where j is the bit index $j=0,....N$. $b_{i,j}$ is the input sequence, $b_{1,j}$ and $b_{2,j}$. x_j is the transmitted code words. y_j is the received code words. Transition metric is set equivalent to a priori LLR $b_{1,j}^a$ or to zero.

Compute Alpha Coefficient: Forward recursion is given to compute the alpha coefficient. After it is accomplished for a particular window, one pair of its corresponding a priori LLRs $b_{1,j}^a$ and $b_{2,j}^a$ is given as input, in the rising order of the bit index j. Here every single α [14, (3)] value is given by

$$\alpha_{j+1}(s') = \max_{s \to s}^* (\alpha_j(s) + \sum_{i=1}^{2} \gamma_{i,j}(s,s') \tag{4}$$

where $s \to s$ present the set of all states s that can be transition into the state s. The forward recursion for the early window is reset independently and for the supplementary windows, it is initialized employing α state

Table 1: Decomposition of max* operation

Op 1	Simultaneously calculate max (p-q) and $\|p$-$q\|$
Op 2	Determine if $\|p$-$q\|> 0.75$
Op 3	Determine if $\|p$-$q\|> 0$ or $\|p$-$q\|>2$ depending on the outcome of operation 2
Op 4	Add max(p-q) to the value selected from the set {0.75,0.5,0.25,0}

of the preceding window. Jocobian logarithm [15] is represented by the max* operation and approximated using LUT [11] for the parameters m and n according to

$$\max^*(m,n) = \max(m,n) + \begin{cases} 0.75, & if\ |m-n| = 0 \\ 0.5, & if\ |m-n| \in \{0.25,0.5,0.75\} \\ 0.25, & if\ |m-n| \in \{1,1.25,1.5,1.75,2\} \\ 0, & otherwise \end{cases} \quad (5)$$

Compute Beta Coefficient: Backward recursion is given to compute the beta coefficient. After it is accomplished for a particular window, one pair of its corresponding a priori LLRs $b^a_{1,j}$ and $b^a_{2,j}$ is given as input, in the descending order of the bit index j. Here every single β [14, (4)] value is given by

$$\beta_{j-1}(s) = \max^*_{s \to s'}(\beta_j(s') + \sum_{i=1}^{2} \gamma_{i,j}(s,s') \quad (6)$$

where $s \to r$ represent the set of all states s that can be transition into the state s.

The backward recursion for the early window is reset independently and for the supplementary windows, it is initialized using a β state of the preceding window.

Compute LLR as Output: To compute the extrinsic LLR value, transition metrics [14, (5)] that correspond to present window are generated and is given by

$$\delta_j(s,s') = \alpha_j(s) + \gamma_{2,j}(s,s') + \beta_j(s') \quad (7)$$

Each extrinsic LLR value in the present window of the sequence $b^e_{1,j}$ is generated finally according to

$$b^e_{1,j} = \max^*_{s^0 \to s'}[\delta_j(s,s')] - \max^*_{s^{10} \to s'}[\delta_j(s,s')] \quad (8)$$

where $s^x \to s'$ is the set of transition that imply $b_{i,j}$ has a binary value x.

Discern that (4), (6)–(8) of the LUT-Log-BCJR algorithm contain merely additions, subtractions and the max* calculation of (5). As every single addition and subtraction constitutes a solitary ACS operation, every single max* calculation can be believed equivalent to four ACS operations, as represented in Table I. This design implements the whole algorithm employing ACS units in parallel, every single of that performs one ACS operation each clock cycle.

Furthermore, the design employs a twin-level list structure. At the early register level, a general purpose register R1, R2 and R3 are paired by ACS unit that are utilized to store intermediate results needed for ACS unit in consecutive clock cycles. The subsequent register level consists of Reg bank 1 and Reg bank 2 for temporary storage of LUT variables.

The Reg bank 1 is used to store a priori LLRs $b^a_{1,j}$ and $b^a_{2,j}$ values and LUT constants of (5). Reg bank 2 stores α, β or γ values as shown in Fig. 2. The conventional architecture employs additional hardware and power during synthesis which imposes high chip area which is overcome by using RNS adder in ACS unit.

Proposed LUT-log-BCJR Architecture: In this system, RNS adder is introduced in ACS unit. The carry free property of RNS adder shortens the critical path and reduces the area, delay and power dissipation of the chips. A low intricacy ACS unit is proposed employing RNS adder of Fig 3, that present one ACS operation for each clock cycle. RNS adder shown in Fig. 4 is three stage circuits. The carry-generate bit g_i, carry-propagate bit p_i and half sum bit h_i are computed by pre processing stage as shown in Fig. 5(a). The carry signals c_i is computed by carry computation unit. Fig. 5(b) shows the computation of carry signals. Sum bits are computed in third unit as shown in Fig. 5(c). This adder maintains high operation speed. Without sacrificing delay, the power and area is considerably decreased using RNS adder.

The operation code O= {O_0, O_1, O_2, O_3, O_4, O_5, O_6} endow the ACS unit control signals, that can be utilized to present the purposes tabulated in Table II. The intermediate results of the registers R_1, R2 and R_3 of Fig. 2 is stored by four operations of max* calculation.

Operation 1: When operation code O=101100 is activated, p takes the value from R1, q from R2 and R3 stores the result |R1-R2|. The result C0 determines max (R1, R2).

Operation 2: When operation code O=110010 is activated, p=0.75 value is taken from register bank 1 and q takes value from R3. The result C1 determines |R1-R2|>0.75 and |R1-R2| is not stored in R3.

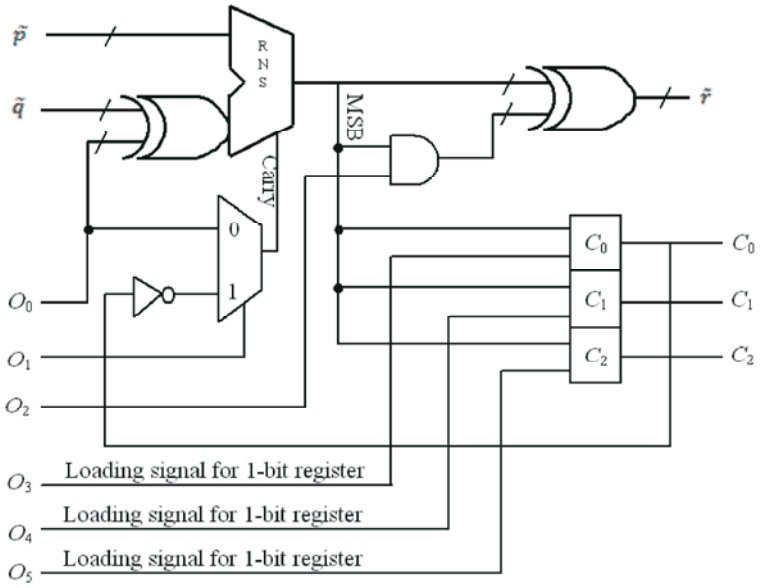


Fig. 3: ACS Unit

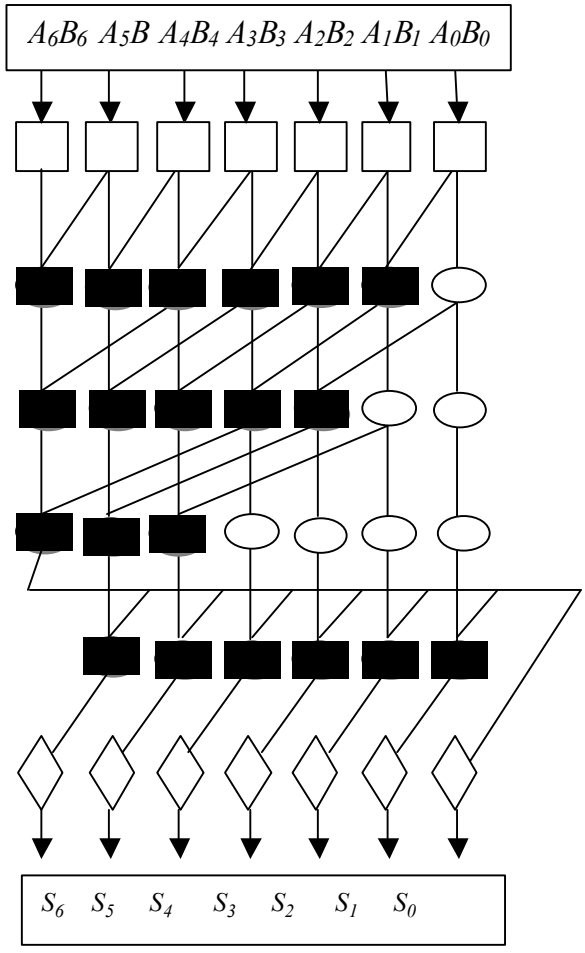
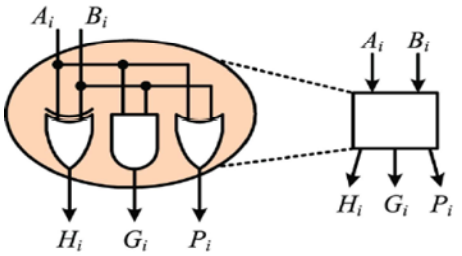
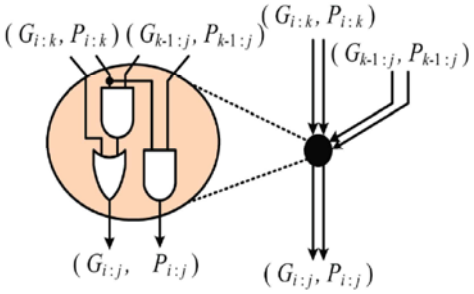


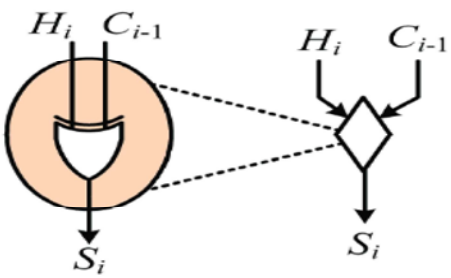
Fig. 4: RNS adder



(a) First stage



(b) Second stage



(c) Third stage

Fig. 5: Logic level implementation of the basic cells

Table II: Operations performed in ACS Unit

O	Function
0000002	$r = p + q$
1000002	$r = p - q$
1011002	$r = \begin{cases} p-q & if\ p \geq q \\ (q-p)-0.25 & if\ p < q \end{cases}$ $C_0 = \begin{cases} 0_2 & if\ r \geq 0 \\ 1_2 & if\ r < 0 \end{cases}$
1100102	$r = \begin{cases} p-q & if\ C_0 = 0_2 \\ (q-p)-0.25 & if\ C_0 = 1_2 \end{cases}$ $C_1 = \begin{cases} 0_2 & if\ r \geq 0 \\ 1_2 & if\ r < 0 \end{cases}$
1100012	$r = \begin{cases} p-q & if\ C_0 = 0_2 \\ (q-p)-0.25 & if\ C_0 = 1_2 \end{cases}$ $C_2 = \begin{cases} 0_2 & if\ r \geq 0 \\ 1_2 & if\ r < 0 \end{cases}$

Operation 3: The result of the test |R1-R2|>0 or |R1-R2|>2 of the test is determined depending on |R1-R2|>0.75. When the operation code O=110001 is activated, p takes the constant value of 0 or 2 and q takes the value stored in R3.

Operation 4: When operation code O=000000 is activated, p value is determined by max (R1, R2) as identified by C0. Reliant on C1 and C2 contents, the set {0.75, 0.50, 0.25, 0} select the operand q value. As a result we have,

$$r = \max(R_1, R_2) + \begin{cases} 0.75 & if\ C_1 = 0_2, C_2 = 0_2 \\ 0.5 & if\ C_1 = 0_2, C_2 = 1_2 \\ 0.25 & if\ C_1 = 1_2, C_2 = 0_2 \\ 0 & if\ C_1 = 1_2, C_2 = 1_2 \end{cases}$$

as required by (5).

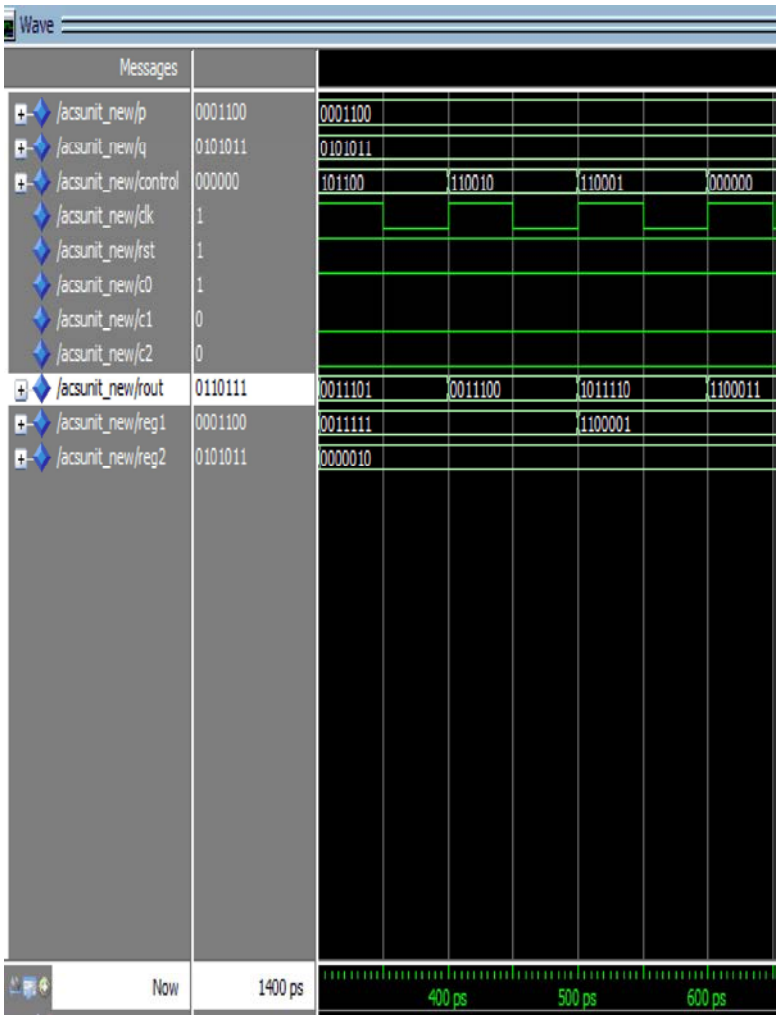


Fig. 6: Simulation result of ACS unit

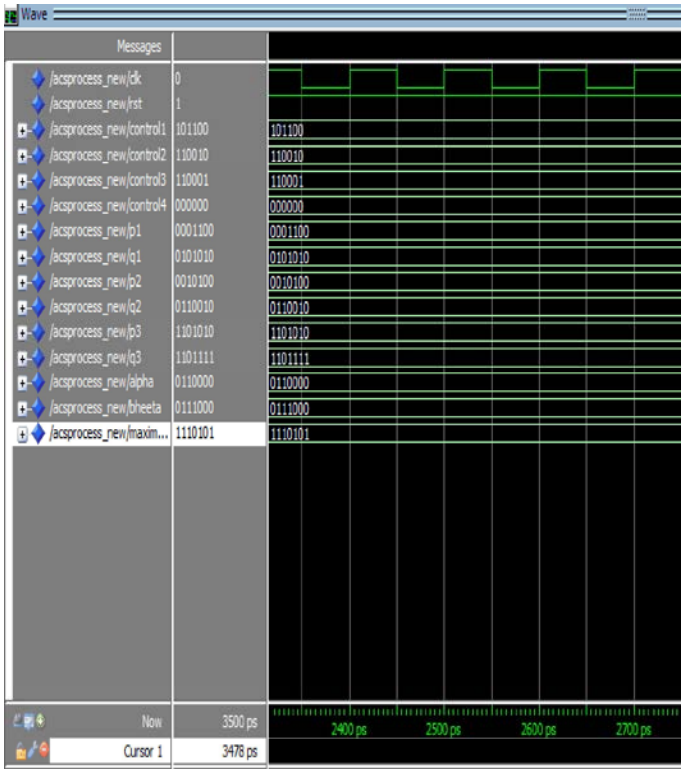


Fig. 7: Simulation result of LUT-Log-BCJR turbo decoder

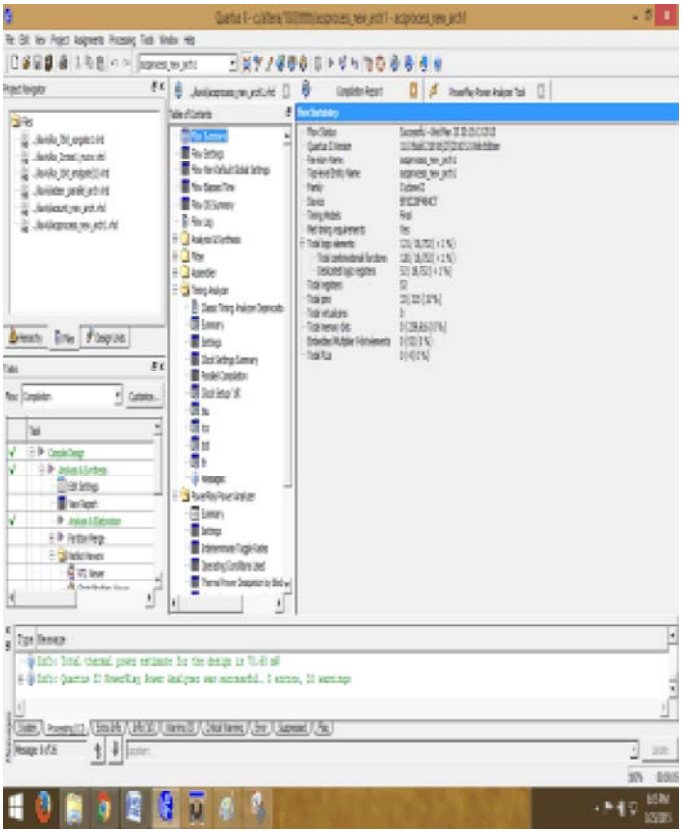


Fig. 8: Synthesis result of area

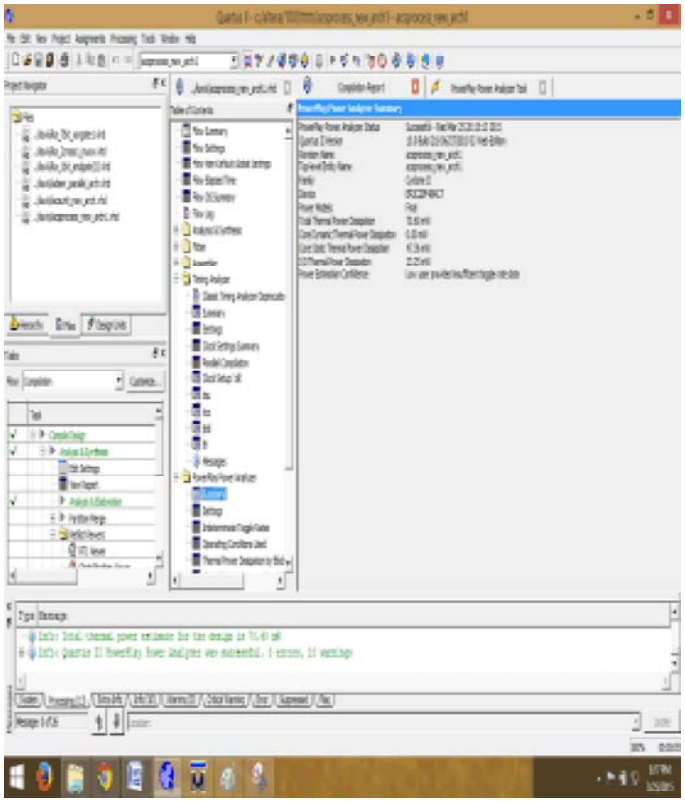


Fig. 9: Synthesis result of power

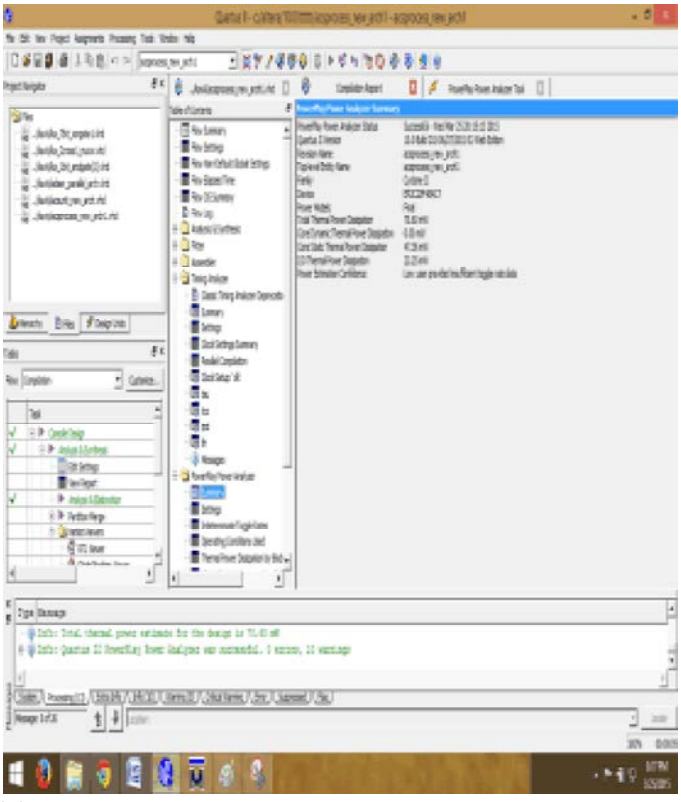


Fig. 10: Synthesis result of delay

Table III: Synthesis Result

Parameter	Percentage
Area	121/18,752 (1%)
Total Thermal Power Dissipation	70.60 mW
Timing Analysis	16.756 ns

Simulation Results: The simulation and synthesis results are summarized for proposed LUT-LOG-BCJR architecture and compared it with other turbo decoder architectures.Turbo decoder architecture using LUT-LOG-BCJR algorithm is simulated using Modelsim 6.4 and synthesis using quartus II tools. The proposed ACS unit is simulated in Fig. 6 and the architecture is simulated in Fig. 7.

ACS unit perform one ACS operation per clock cycle. Here, ACS unit is used to calculate max* operation which constitute four ACS operation. The control signals of the ACS unit are provided by the operation code O= {O0, O1, O2, O3, O4, O5}, which can be used to perform the following functions. During the first clock cycle, operation code O = 101100_2 is activated. The output is the difference of p and q if p > q. C0 is equal to 0_2 if p > q, otherwise C0 = 1_2. During the second clock cycle, operation code O = 110010_2 is activated. The output is the difference of p and q if C0 = 0_2. Otherwise the output rout is the difference of p, q, 0.25 (rout= | q - p - 0.25 |) respectively. C1 is equal to 0_2 if rout > 0, otherwise C1 = 1_2.

During the third clock cycle, operation code O = $110010_{2\ is}$ activated. The output is the difference of p and q if C0 = 0_2. Otherwise the output rout is the difference of p, q, 0.25 (rout= |q - p - 0.25|) respectively. C2 is equal to 0_2 if rout > 0, otherwise C2 = 1_2The operation code O = 000000_2 is activated in the fourth clock cycle which completes the max* calculation as shown in Fig 7. The simulation results show that the hardware intricacy of the proposed design is low. It analyses and estimates the delay, power and area.

Area analyser of Fig. 8 the total logic elements used that includes total combinational functions and dedicated logic registers and gives the percentage of area.

The power analyser of Fig. 9 shows that proposed architecture consumes power of about 70.60 mW. The timing analysis of fig.10 shows that the delay is 16.756 ns. The synthesis results are shown in Table III.

CONCULSION

The proposed LUT-Log-BCJR architecture uses RNS adder in ACS unit. A valid decrease in delay, power and area has been achieved by introducing RNS adder. This technique supports carry free additions and shortens the critical path. Thus the resulting performance gained through RNS adder is substantial.

REFERENCES

1. Liang Li, Robert G. Maunder, Bashir M. Al-Hashimi and Lajos Hanzo, 2013. A Low-Complexity Turbo Decoder Architecture for Energy-Efficient Wireless Sensor Networks, IEEE transactions on very large scale integration (vlsi) systems, 21(1): 14-19.
2. May, M., T. Ilnseher, N. Wehn and W. Raab, 2010. A 150 Mbit/s 3GPP LTE turbo code decoder," in Proc. Design, Autom. Test in Euro. Conf. Exhib. (DATE), pp: 1420-1425.
3. Studer, C., C. Benkeser, S. Belfanti and Q. Huang, 2011. Design and implementation of a parallel turbo-decoder ASIC for 3GPP LTE, IEEE J. Solid-State Circuits, 46: 8-17.
4. Wong, C., Y. Lee and H. Chang, 2009. A 188-size 2.1 mm reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in Proc. Symp. VLSI Circuits, pp: 288-289.
5. Robertson, P., P. Hoeher and E. Villebrun, 1997. Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding, Euro. Trans. Telecommun., 8(2): 119-125.
6. Bickerstaff, M.A., D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L.M. Davis, G. Woodward, C. Nicol and R.H. Yan, 2002. A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18- m CMOS, IEEE J. Solid-State Circuits, 37(11): 1555-1564.
7. Bickerstaff, M., L. Davis, C. Thomas, D. Garrett and C. Nicol, 2003. A 24 Mb/s radix-4 log-MAP turbo decoder for 3GPP-HSDPA mobile wireless,"in Proc. IEEE Int. Solid-State Circuits Conf., pp: 150-484.
8. Li, F.M., C.H. Lin and A.Y. Wu, 2008. Unified convolutional/turbo decoder design using tile-based timing analysis of VA/MAP kernel, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 16(10): 1063-8210.
9. Hanzo, L., T.H. Liew, B.L. Yeap, R. Tee and S.X. Ng, 2011. Turbo Coding, Turbo Equalisation and Space-Time Coding. New York:Wiley.
10. Hanzo, L., J.P. Woodard and P. Robertson, 2007. Turbo decoding and detection for wireless applications, Proc. IEEE, 95(6): 1178-1200.
11. Berrou, C., A. Glavieux and P. Thitimajshima, 1993. Near Shannon limit error correcting coding and decoding: Turbo codes, in Proc. IEEE Int. Conf. Commun., pp: 1064-1070.

12. Robertson, P., E. Villebrun and P. Hoeher, 1995. A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain, in Proc. IEEE Int. Conf. Commun., pp: 1009-1013.
13. Schurgers, C., F. Catthoor and M. Engels, 2001. Memory optimization of MAP turbo decoder algorithms, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 9(2): 305-312.
14. Masera, G., M. Mazza, G. Piccinini, F. Viglione and M. Zamboni, 2002. Architectural strategies for low-power VLSI turbo decoders, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 10(3): 279-285.
15. Wu, C.M., M.D. Shieh, C.H. Wu, Y.T. Hwang and J.H. Chen, 2005. VLSI architectural design tradeoffs for sliding-window log-MAP decoders, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., 13(4): 439-447.