VLSI Implementation of Turbo Decoder Architecture for WSN

Sohail Siraj. S
PG Scholar, Dept of ECE
SMVEC, Puducherry, India

Senthil. P
Assistant Professor, Dept of ECE
SMVEC, Puducherry, India

***Abstract*— This Turbo codes have recently been considered for energy-efficient wireless sensor networks, since they facilitate low transmission energy consumption. To reduce the overall energy consumption of wireless sensor networks Look-Up-Table-Logarithmic-BCJR (LUT-Log-BCJR) architectures having low processing energy consumption are required. This work, decomposes the LUT-Log-BCJR architecture into its most fundamental Add Compare Select (ACS) operations and performed using a novel low-complexity ACS unit. The proposed architecture employs an order of magnitude fewer gates than the most recent LUT-Log-BCJR architectures, providing a good result in energy consumption reduction, lower chip area and lower delay in processing All the experiments are performed and simulated in XILINX ISE 13.2 environment using VHDL(Very high speed integrated circuit hardware description language).**

Keywords – LOG-LUT BCJR, Turbo codes, ACS(Add compare select), WSN.

I. NTRODUCTION

Turbo codes are a class of high-performance forward error correction (FEC) codes developed in 1993, which were the first practical codes to closely approach the channel capacity, a theoretical maximum for the code rate at which reliable communication is still possible given a specific noise level[1]. Turbo codes have been used also for its high coding gain. Wireless sensor networks are considered to be energy efficient because they rely on batteries that are light and inexpensive and can operate on extended periods of time. The WSN's energy consumption is mainly dominated by the transmission energy [2],[3].Turbo codes have recently found application in these scenarios [3], [4], since their high coding gain facilitates reliable communication when using a reduced transmission energy. However the reduced transmission energy is counteracted by the turbo decoder's energy consumption[4].For this reason Turbo codes designed for energy efficient WSNs must reduce the overall energy consumption i.e Sensor's transmission energy and Turbo decoder's energy consumption.

The BCJR algorithm is an algorithm for maximum a posterior decoding of error correcting codes defined on trellises (principally convolutional codes). The algorithm is named after its inventors: Bahl, Cocke, Jelinek and Raviv. This algorithm is critical to modern iteratively-decoded error-correcting codes including turbo codes and low-density parity-check codes. The BCJR algorithm contains multiplications and divisions in computation which is complex and due to this reason it was not used for almost 30 years. Since the invention of turbo codes BCJR algorithm have been started to be used. Now many variants have been evolved from BCJR algorithm. They are

- MAX-LOG BCJR ALGORITHM
- LOG-LUT BCJR ALGORITHM

All the computations when being transferred into the logarithmic domain becomes additions and subtractions which reduce the complexity. The Max-Log-BCJR algorithm appears to lend itself to both high-throughput scenarios, as well as to the above mentioned energy-constrained scenarios. This is because low turbo decoder energy consumption is implied by Max-Log-BCJR algorithm's low complexity. However, this is achieved at the cost of degrading the coding gain by 0.5 dB compared to the optimal Log-BCJR algorithm.

IJERT

This motivates the employment of the Look-Up-Table-Log-BCJR (LUT-Log-BCJR) algorithm [5] in energy-constrained scenarios, since it approximates the optimal Log-BCJR more closely than the Max-Log-BCJR and therefore does not suffer from the associated coding gain degradation.However,to the best of our knowledge, no LUT-LOG-BCJR ASICS have been specifically designed for energy-constrained scenarios. Previous LUT-Log-BCJR turbo decoder designs [6]–[9] were developed as a part of the on-going drive for higher and higher processing throughputs, although their throughputs have since been eclipsed by the Max-Log-BCJR architectures. This opens the door for a new generation of LUT-Log-BCJR ASICs that exchange processing throughput for energy efficiency.

II. TURBO ENCODER AND DECODER SCHEMATIC

As shown in figure Turbo Encoder is a parallel concatenation of two recursive convolutional encoders separated by an interleaver. The Turbo Encoder [10] produces three outputs (one systematic output bit and two parity bits) for every input bit.

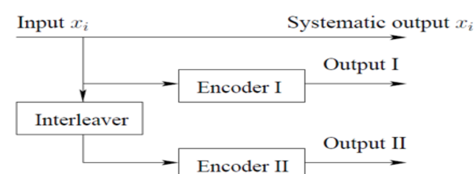

Fig 1.Turbo Encoder

The Turbo decoder [10] is shown as in Figure 2 .The decoder of Figure 2 operates iteratively, and in the first

iteration the first component decoder takes channel output values only, and produces a soft output as its estimate of the data bits. The soft output from the first encoder is then used as additional information for the second decoder, which uses this information along with the channel outputs to calculate its estimate of the data bits. Now the second iteration can begin, and the first decoder decodes the channel outputs again, but now with additional information about the value of the input bits provided by the output of the second decoder in the first iteration. This additional information allows the first decoder to obtain a more accurate set of soft outputs, which are then used by the second decoder as a-priori information. This cycle is repeated, and with every iteration the Bit Error Rate (BER) of the decoded bits tends to fall. However the improvement in performance obtained with increasing numbers of iterations decreases as the number of iterations increases. Hence, for complexity reasons, usually only about 8 iterations are used.

Due to the interleaving used at the encoder, care must be taken to properly interleave and de-interleave the LLRs which are used to represent the soft values of the bits, as seen in Figure 4.3. Furthermore, because of the iterative nature of the decoding, care must be taken not to re-use the same information more than once at each decoding step.

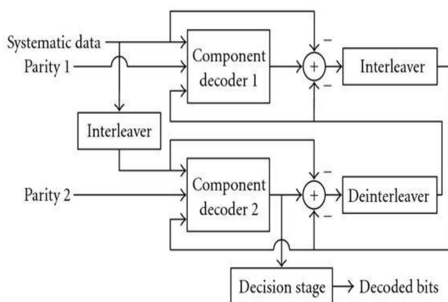

Fig 2.Turbo Decoder

III. FORWARD AND BACKWARD RECURSIONS

Log-BCJR algorithm [11] is composed of the following four parts.

The values of γ depend on the inputs of the convolutional decoder. There are two inputs, the encoded LLRs input and the uncoded LLRs input. As shown in Figure 2, the encoded LLRs input is the LLRs of the encoded sequence received from the channel $\widehat{c_n^c}$. The uncoded LLRs input is $\widehat{y^a}$. For a transition T, the γ_y and γ_c can be calculated as

$$\gamma_y(T) = \left(1 - y(T)\right)\widehat{y^a}_{n(T)} \tag{1}$$

$$\gamma_c(T) = \left(1 - c(T)\right)\widehat{c^c}_{n(T)} . \tag{2}$$

The values of α depend on the γ values and α values from the previous step in the trellis. Hence, it requires a forward recursion in the trellis to obtain all the α values. For a state S, in step n, the function to calculate α is:

$$\alpha(S) = \max_{T\epsilon to\,(S)}^{*}\left(\gamma_y(T) + \gamma_c(T) + \alpha(fr(T))\right) . \tag{3}$$

Where $\alpha(S_1) = 0$.

The values of β depend on the γ values and β values from the next step in the trellis. Hence, it requires a backward recursion in the trellis to obtain all the β values. For a state S, in step n, the function to calculate β is:

$$\beta(S) = \max_{T\epsilon fr\,(S)}^{*}\left(\gamma_y(T) + \gamma_c(T) + \beta(to(T))\right). \tag{4}$$

Where $\beta(S_{n-1}) = 0$

4. δ_y calculation: The values of δ_y can be calculated according to

$$\delta_y(T) = \gamma_c(T) + \alpha\left(from(T)\right) + \beta\left(to(T)\right) \tag{5}$$

5. Finally, the extrinsic information can be calculated based on δ values. The extrinsic LLRs of the uncoded bits $\widehat{y_e}$ are:

$$\widehat{y_n^e} = \max_{T|y(T)=0}^{*}(\delta_y(T)) - \max_{T|y(T)=1}^{*}(\delta_y(T)). \tag{6}$$

IV. PROPOSED WORK

The proposed energy-efficient LUT-Log-BCJR architecture[12] is shown in Figure 3. Unlike conventional architectures, it does not use separate dedicated hardware for the three recursions shown in Section 3. Instead, our architecture implements the entire algorithm using 2^m ACS units in parallel, each of which performs one ACS operation per clock cycle.

The Main Memory consists of all the processed a priori and uncoded LLRS required for the processing of the entire LOG LUT BCJR ALGORITHM. The Main memory is designed as a fixed point data (5-bit integer part and 2-bit fractional part) RAM. The LLRS are being fed into the main memory using “mem in” operation and the LLRS are read out using the “mem out” operation. Now the LLRS to be processed are read out from the main memory are fed into the ACS units which are arranged in a parallel manner. 2^m ACS units are used in our architecture,in our work we use memory elements ,m=2,so 4 ACS units are used.

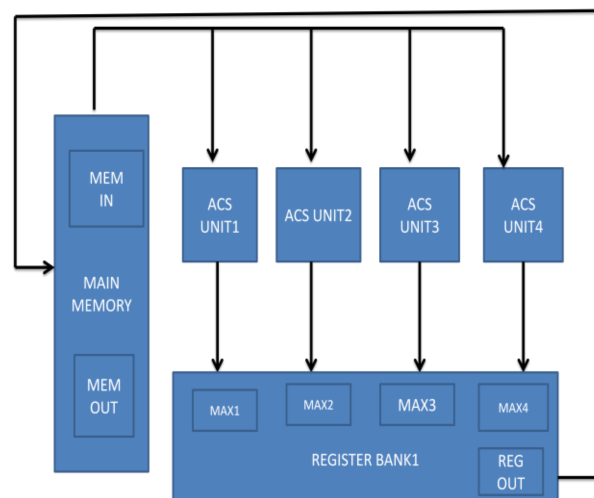

Figure 3. Proposed Architecture

The ACS units performs addition and subtraction in one clock cycle and max^* operation in four clock cycles. So all

IJERT

the max^* operations for all the 4 states $STATE_0, STATE_1, STATE_3, STATE_4$ are implemented in a total of 4 clock cycles whereas in the conventional architecture it takes 16 clock cycles for a particular window.

The lut constants described in the equation are stored in the Register bank. The Register bank is read out and written in using the "reg in" and "reg out" operation as shown in the figure. So during the max^* operation for utilising the lut constants, register bank is used. The intermediate results used in the max^* operaion are also stored in the register bank, so whenever needed it is read out from the register bank and processed using the ACS units. Since the proposed architecture supports a fully parallel arrangement of an arbitrary number of ACS units of Figure, it may be readily applied to any LUT-Log-BCJR decoder, regardless of the specific convolutional encoder parameters employed.

The Proposed architecture can be readily applied to any type of decoding algorithm such as viterbi, max-log-map algorithm, since the ACS units are low complexity functional units which operate based on the operation signals represented by O. Now Let us now discuss the Operations of the ACS unit.

V. DECOMPOSITION OF LOG-LUT BCJR ALGORITHM

Equations (1), (2), (3), (4), (5), (6) of the LUT-Log-BCJR algorithm comprise only additions, subtractions and the max* calculation of (12,Equation-2). While each addition and subtraction constitutes a single ACS operation, each max* calculation can be considered equivalent to four ACS operations. Now the max* operation is implemented in four steps.The MAX-LOG-BCJR algorithm does not use the approximation it simply finds out the maximum of two LLRS,so it only uses one ACS operation after the calculation of alpha or beta state metric Similarly, fewer ACS operations are required, when employing the Constant- Log-BCJR [13] algorithm. These alternative algorithms reduce the hardware complexity and increase the throughput, therefore reducing the decoder's energy consumption. However, this is achieved at the cost of requiring higher transmission energy to achieve the same BER performance as the LOG LUT BCJR algorithm.

TABLE 1. DECOMPOSITION OF MAX *

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

VI. ACS UNITS

These are low complexity functional units which are collectively capable of performing the entire LOG-LUT BCJR algorithm. ACS units[13] form the core unit in viterbi decoder, the architecture might vary for both the algorithms, but these are low complexity units which occupy less chip area and perform with less delay.

The proposed work employs ACS units in parallel, the critical path delay is re Further wastage is avoided, since the critical paths of our functional units are naturally short- and equally-lengthed, eliminating the requirement for additional hardware to manage them. In this section we propose the novel low-gate-count ACS unit of Figure 4, which performs one ACS operation per clock cycle.

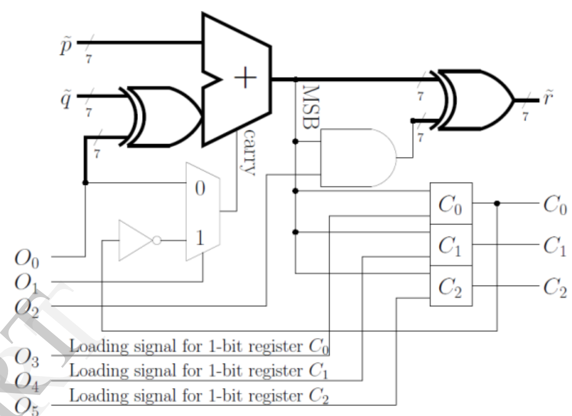

Figure 4. ACS Unit

All the ACS operations are performed in 2's complement fixed point representation. As shown in the above figure inputs and outputs are represented in 7-bit fixed point representation.

A. Operations of ACS Unit

The control signals of the ACS unit are provided by the operation code O={$O_0, O_1, O_2, O_3, O_4, O_5$} which can be used to perform the functions listed in Table The addition and subtraction operations for the equations can be simply performed by the operation code O= "000000" and O="100000" respectively. The max* operation which requires four steps as exemplified in the table are implemented in four ACS operations as listed below.

- Operation 1(O="101100"):

In this clock cycle the max* calculation is activated by using the operation code O = "101100", now the operands for max* operation p and q are read out from the main memory and fed into the respective ACS units, then the result r is fed into the register bank. The result r is then stored in register bank, which is the approximated as $|p - q|$. The result C0 determines max (p,q).

IJERT

- Operation 2(O="110010"):

The LUT comparison performed during the second ACS operation is activated by the operation code O = "110010" of Table. Operand $\hat{p}$ uses the constant decimal value 0.75 which is provided by the register bank in the architecture and $\hat{q}$ takes the maximum of $\hat{p}$ and $\hat{q}$ which was computed in the previous clock cycle. The value for $\hat{q}$ is read out from the register bank. In this clock cycle, the result $\hat{r}$ is not stored, while the result stored in C_1 provides the outcome of the test $|\hat{p} - \hat{q}| > 0.75$, as required by the second ACS operation described in Table 2

.

- Operation 3(O="110001"):

Similarly to the previous clock cycle, the result of the test $|\hat{p} - \hat{q}| > 0$ or of the test $|\hat{p} - \hat{q}| > 2$ is determined depending on whether it was previously decided that $|\hat{p} - \hat{q}| > 0.75$, now we employ the operation code O = 110001 of Table 2,for $\hat{q}$ again use the maximum of $\hat{p}$ and $\hat{q}$ which was stored in the register bank and for $\hat{p}$ substitute the constant value of 0 or 2, as appropriate. As shown in Equation (2), these constant values are the first and third entries of the LUT.

- Operation 4(O="000000"):

The max* calculation of Equation (2) is completed in the fourth clock cycle by using the operation code O = 000000 of Table II.Now for operand of $\hat{p}$ use the maximum of of $\hat{p}$ and $\hat{q}$

TABLE 2.OPERATIONS OF ACS UNIT

O	Function
000000	$\hat{r} = \hat{p} + \hat{q}$
100000	$\hat{r} = \hat{p} - \hat{q}$
101100	$\hat{r} = \begin{cases} \hat{p} - \hat{q} & if\ \hat{p} \geq \hat{q} \\ (\hat{q} - \hat{p}) - 0.25 & if\ \hat{p} < \hat{q} \end{cases}$ $C_0 = \begin{cases} 0\ if\ \hat{p} \geq \hat{q} \\ 1\ if\ \hat{p} < \hat{q} \end{cases}$
110010	$\hat{r} = \begin{cases} \hat{p} - \hat{q} & if\ C_0 = 0 \\ (\hat{p} - \hat{q}) - 0.25 & if\ C_0 = 1 \end{cases}$ $C_1 = \begin{cases} 0\ if\ \hat{r} \geq 0 \\ 1\ if\ \hat{r} < 0 \end{cases}$
110001	$\hat{r} = \begin{cases} \hat{p} - \hat{q} & if\ C_0 = 0 \\ (\hat{p} - \hat{q}) - 0.25 & if\ C_0 = 1 \end{cases}$ $C_2 = \begin{cases} 0\ if\ \hat{r} \geq 0 \\ 1\ if\ \hat{r} < 0 \end{cases}$

as identified by C_0 stored in the register bank and for $\hat{q}$, it is selected from the set {0.75, 0.5, 0.25, 0} depending on the contents of C1 and C2 of Figure 4.As a result, we have

$$max^*(\hat{p}, \hat{q}) \approx \max(\hat{p}, \hat{q}) + \begin{cases} 0.75\ if\ C_1 = 0, C_2 = 0 \\ 0.5\ \ if\ C_1 = 0, C_2 = 1 \\ 0.25\ if\ C_1 = 1, C_2 = 0 \\ 0\ \ if\ C_1 = 1, C_2 = 1 \end{cases}$$

VII. IMPLEMENTATION RESULTS

The Turbo Decoder is synthesized and simulated using XILINX ISE 13.2.The RTL schematic and simulation wave window is given here.

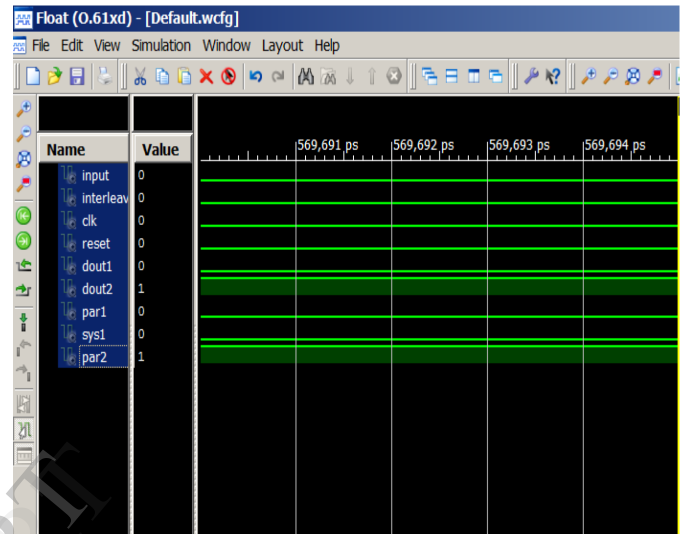

Figure 4.Turbo Encoder wave window

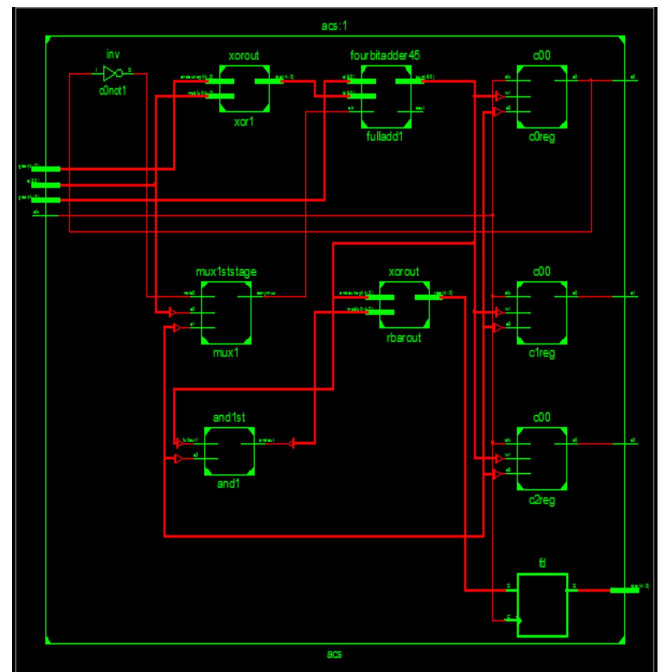

Figure 5 .ACS Unit RTL Schematic

IJERT

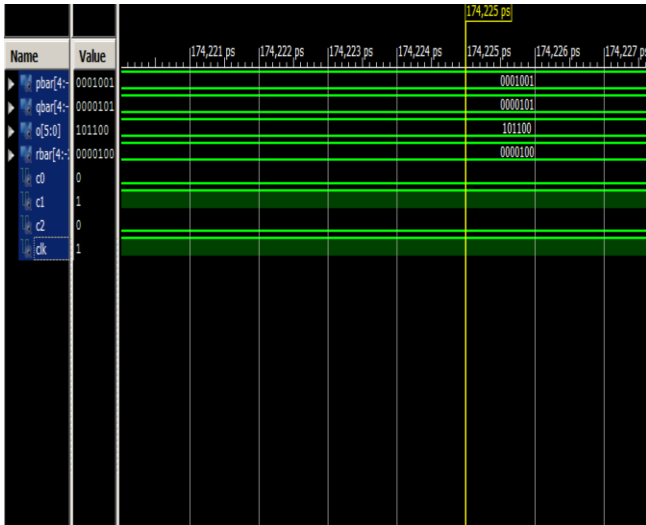

Fig 6 ACS Unit Wave Window

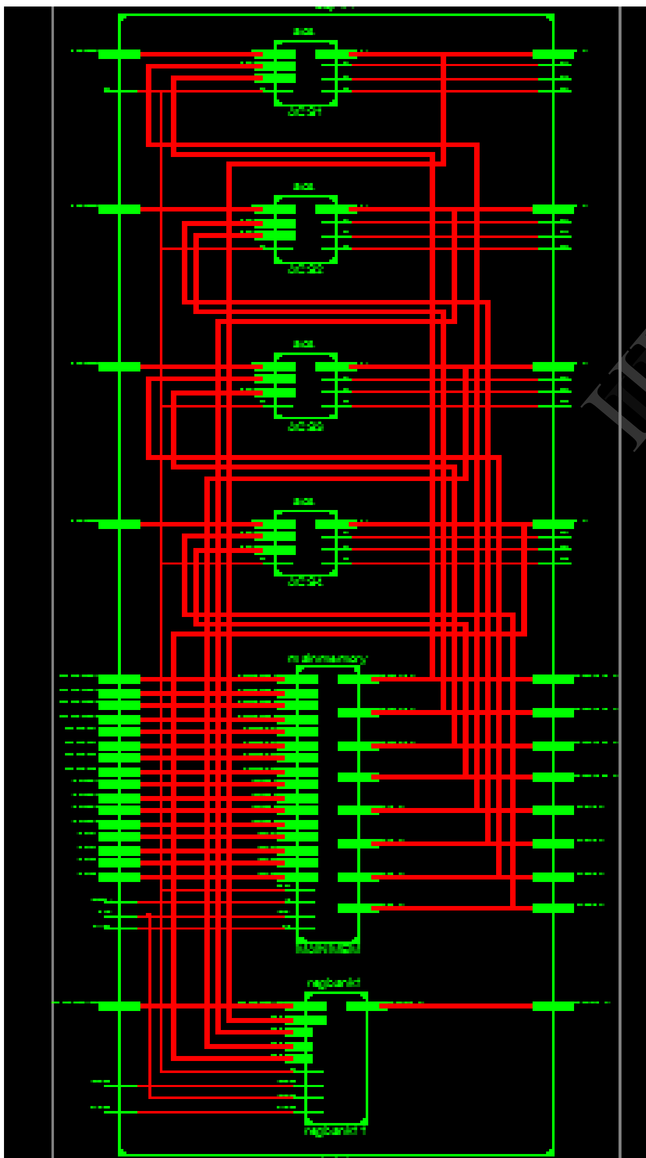

Fig 7 Proposed Architecture RTL Schematic

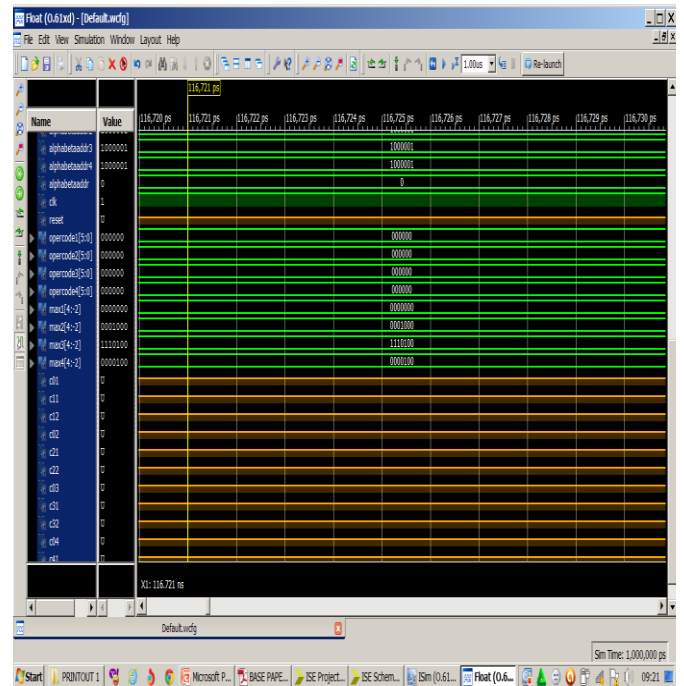

Figure 8 Proposed Architecture Wave Window

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	7	93120	0%
Number of Slice LUTs	16	46560	0%
Number of fully used LUT-FF pairs	7	16	43%
Number of bonded IOBs	25	360	6%
Number of BUFG/BUFGCTRLs	1	32	3%

Figure 9. Device Utilization Summary

IJERT

CONCLUSION

BCJR algorithm has been ignored for the past 30 years due to its high complexity in computation .Log BCJR algorithm due to its reduced complexity have found interest in turbo codes. The proposed architecture employs fewer magnitude ACS units which implements the entire LOG-BCJR algorithm in parallel. The work has been implemented in Virtex-6 FPGA and simulated in Xilinx ISE 13.2 and its results have been computed. Results show the reduced delay in processing and low hardware complexity. A further idea for improving the design would be to divide the BCJR trellis into a number of windows that can be processed in parallel using additional ACS units.

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes," in Proceedings of the IEEE International Conference on Communications, vol. 2, Geneva, Switzerland, 1993, pp. 1064–1070.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 52, pp. 292–422, 2008.

[3] P. Corke, T. Wark, R. Jurdak, H. Wen, P. Valencia, and D. Moore, "Environmental Wireless Sensor Networks," Proceedings of the IEEE, vol. 98, no. 11, pp. 1903–1917, 2010.

[4] L. Li, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "An energyefficient error correction scheme for IEEE 802.15.4 wireless sensor networks," Transactions on Circuits and Systems II, vol. 57, no. 3, pp. 233–237, 2010.

[5] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and Sub-Optimal Maximum A Posteriori Algorithms Suitable for Turbo Decoding," European Transactions on Telecommunications, vol. 8, no. 2, pp. 119–125, 1997.

[6] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R.-H. Yan, "A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18-_m CMOS," IEEE Journal of Solid-State Circuits, vol. 37, no. 11, pp.1555–1564, 2002.

[7] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 Log-MAP turbo decoder for 3GPP-HSDPA mobile wireless," in IEEE International Solid-State Circuits Conference, vol. 1, 2003, pp. 150–484.

[8] Z. Wang, "High-Speed Recursion Architectures for MAP-Based Turbo Decoders," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15, no. 4, pp. 470–474, 2007.

[9] F.-M. Li, C.-H. Lin, and A.-Y. Wu, "Unified Convolutional/Turbo Decoder Design Using Tile-Based Timing Analysis of VA/MAP Kernel," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 10, pp. 1063–8210, 2008.

[10] L. Hanzo, T. H. Liew, B. L. Yeap, R. Tee, and S. X. Ng, Turbo Coding, Turbo Equalisation and Space-Time Coding. John Wiley & Sons Inc, 2011.

[11] C. M. Wu, M. D. Shieh, C. H. Wu, Y. T. Hwang, and J. H. Chen, "VLSI Architectural Design Tradeoffs for Sliding-Window Log-MAP Decoders," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 4, pp. 439–447, 2005.

[12] A LowComplexity Turbo Decoder Architecture for Energy-Efficient Wireless Sensor Networks ,IEEE Transactions on Very Large Scale Integration (VLSI) Systems Volume: 21 , Issue: 1 ,2013

[13] M. C. Valenti and J. Sun, "The UMTS turbo Code and an Efficient Decoder Implementation Suitable for Software-Defined Radios," International Journal of Wireless Information Networks, vol. 8, no. 4, pp. 203–215, 2001.

IJERT