An Energy-Efficient Lut-Log-Bcjr Architecture Using Constant Log Bcjr Algorithm

Nidhu Devi M*, Seena George**
*(M.tech scholar, Department of ECE, Sree Narayana Gurukulam College of Engineering, Kolenchery, India)
**(Asst. Professor, Department of ECE, Sree Narayana Gurukulam College of Engineering, Kolenchery, India)

Abstract
Error correcting codes are used to correct the data from the corrupted signal due to noise and interference. There are many error correcting codes. Among them turbo codes is considered to be the best because it is very close to the Shannon theoretical limit. The MAP algorithm is commonly used in the turbo decoder. Among the different versions of the MAP algorithm Constant log BCJR algorithm have less complexity and good error performance. The Constant log BCJR algorithm can be easily designed using look up table which reduces the memory consumption. The proposed Constant log BCJR decoder is designed to decode two blocks of data at a time, this increases the throughput. The complexity of the decoder is further reduced by the use of the add compare select (ACS) units and registers. The proposed decoder is simulated using Xilinx ISE and synthesized using Sparten3 FPGA and found out that Constant log BCJR decoder utilized less amount of memory and power than the LUT log BCJR decoder.

Index Terms—Add compare select(ACS), turbo codes, Constant log BCJR Algorithm.

I. INTRODUCTION
Over the years, there has been increase in the use of digital communication in the field of computer, cellular and satellite communication. The digital communication, the information is the binary data which is modulated on the analog waveform and transmitted through the communication channel. In the communication channel the information may be interfered by the noise[7]. At the receiver station the corrupted signal is demodulated and then it is converted in the binary bits. Due to the noise the data may not be the correct replica of the transmitted signal. So we use channel coding[1] scheme to protect against the noise and interference.

The turbo codes[2] have been used in many standard communications. Such as the standard used by the third Generation Partnership Project (3GPP), Digital Video Broadcasting (DVB), Universal Mobile Telecommunication Systems (UMTS), NASA Consultative Committee for Space Data System (CCSDS),Wireless Local Area Network (WLAN), Global System for Mobile communication (GSM) and Advanced Television Committee(ATSC) standard. The turbo codes are the concatenated versions of the two or more codes. Decoding is done by using LLR calculation. The iterative nature of the turbo decoding algorithm increases the complexity. The two decoding algorithm used for the turbo codes are Maximum A Posteriori (MAP) and Soft Output Viterbi Algorithm (SOVA) [3]. The MAP algorithm[4],[5] is based on the most likely data sequence. While SOVA is based on the most likely connected path through the trellis tree. The MAP algorithm performed well than the SOVA at low SNR. However the implementation of the decoder using the MAP algorithm is more difficult.

In order to reduce the computational complexity of the MAP algorithm is usually implemented in the logarithmic domain and corresponding algorithm is the log-MAP algorithm. Further the log-MAP algorithm can be simplified. Then we get max-log-MAP, constant log MAP and linear log map algorithm. The log MAP algorithm can be implemented using look up table. The LUT-log-BCJR[6] decoder is based on this. By the use of the constant log-BCJR(constant log-MAP) algorithm we can reduce the look up table elements than in the LUT-log- BCJR[8] algorithm. Again the complexity in the hardware can reduce by the use of low complex structure. So new architecture is developed which contain only registers and add compare and select unit for the constant log-BCJR decoder block. Also the decoder can decode two blocks of data at a time. This increase the throughput of the system.

Many algorithms are introduced to perform decoding in a faster way. In this paper, we are introducing a new area reducing architecture for turbodecoder. The reminder of this paper is organized as follows: Section II briefly review our objective. Section III, briefly reviews previous algorithms and their architectures for performing decoding. In section IV, discuss the new area efficient turbodecoder architecture. Section V gives the
A detailed comparison between the proposed method and the previous method. Finally, we concluded this paper in section VI.

II. OBJECTIVE

Design of turbo decoder using Constant log BCJR algorithm and analyze its performance.

III. PREVIOUS METHODOLOGY

Turbo encoder[9] comprises a parallel concatenation of two convolutional encoders. LUT-LOG-BCJR decoder processes Logarithmic Likelihood Ratios (LLR). Each encoder converts an unencoded bit sequence into corresponding encoded bit sequence. Fig 1 depicts a turbo decoder which comprises a parallel concatenation of two decoders that employ LUT-LOG-BCJR algorithm. Where each LLR = ln (p(b=0))/p(b=1).

The reverse state metrics can be calculated as follows:

\[
\beta_{j-1}(s) = \max_{s' \rightarrow s} \left( \beta_j(s') + \sum_{i=1}^{2} y_{i,j}(s, s') \right)
\]  

(3)

The conventional architecture generates one extrinsic LLR per clock cycle. Therefore it achieves a high throughput, provided that it can be operated at a high clock frequency. However, the recursions involve calculations that must be performed in series. Therefore, conventional architecture requires additional hardware. In summary, conventional architecture achieves high throughput.

IV. PROPOSED METHODOLOGY

4.1 TURBO ENCODER:

Mainly turbo encoding[11] is based on the parallel concatenation of two Recursive Systematic Convolutional encoders (RSC) given in fig.1. Two encoders produces the redundant data as a parity bits (X_kp1, X_kp2). The input data stream and parity bits are combined in series to form the turbo coded word. The interleaver separating two RSC encoders prevents at least one of the encoders to terminate quickly.

\[
\text{Max}^*(p,q) = \begin{cases} 
0.75, & \text{if } (p-q) = 0 \\
0.5, & \text{if } (p-q) \in (0.25, 0.5, 0.75) \\
0.25, & \text{if } (p-q) \in (1, 1.25, 1.5, 1.75) \\
0, & \text{otherwise}
\end{cases}
\]  

(2)

The max*[10] operation is used to represent the Jacobian logarithm, which may be approximated using a LUT for the parameters p and q according to

\[
\gamma_{k,m} = \pi_k \exp \left( x_k^p v_k^p + y_k^p v_k^p \right)
\]  

(4)
One of the main reasons of the turbo codes is that they produce high weight code words. For example, if the input sequence is low weight code, then the systematic and parity1 output may produce a low weight codes. Because of the interleaver the parity 2 less likely to produce low weight code word. The interleaver shuffle the input sequence in such a way that when introduce to the second encoder, it is more likely to produce a high weight codeword. This is the ideal for the code because high weight code is result in better decoder performance. Intuitively, when one of the decoder produces a weak codeword, the other encoder has the least probability of producing weak code word because of the interleaver. The concatenated version of the two codeword is therefore a strong code word. The interleaver design also affects the turbo decoder performance by reducing the degree of correlation between the soft-output of each decoder which becomes the extrinsic information to the other decoder. As the degree of correlation between these two soft information decreases the performance of the turbo decoder increases.

4.2 LUT-LOG-BCJR ARCHITECTURE

In this section we propose a novel LUT-LOG-BCJR architecture for energy constrained scenarios which avoids the wastage of energy. Our philosophy is to redesign the timing of the conventional architecture in a manner that allows its components to be efficiently merged. This produces an architecture comprising only a low-complexity functional units. Which are capable of performing entire LUT-LOG-BCJR algorithm. Furthermore, our approach naturally results in a low area and high clock frequency, which implies low static energy consumption. LUT-LOG-BCJR algorithm can be decomposed into classic ACS operations.

4.3 DECOMPOSITION OF LUT-LOG-BCJR ALGORITHM

LUT-LOG-BCJR algorithm comprises only addition, subtraction and max* operations. Each addition and subtraction constitutes a single ACS operation and each max* calculation can be considered equivalent to four ACS operations. In the general case, when $z > 0$ fraction bits are employed in the twos complement fixed-point LLR representation, a total of $(z+2)$ ACS operations are required to carry out the max* calculation. By contrast only a single ACS operation is required when $z=0$ or when employing the Max-Log-BCJR algorithm, which approximate max* by the max operation. Similarly fewer ACS operation are required when employing the Constant Log BCJR algorithm. This alternative algorithm reduces hardware complexity and increase the throughput, and therefore reducing their energy consumption.

4.4 PROPOSED CONSTANT LOG BCJR ARCHITECTURE

The proposed architecture is shown in fig.3. Unlike the conventional architecture, it does not use the additional hardware for three recursions. Instead our architecture implement the entire algorithm using $2^m$ ACS units in parallel, each of which performs one ACS operation per clock cycle. The proposed architecture ie, SISO[11] block consists of twin-level register structure to minimize highly energy consuming main memory access operations. The first register level comprises R1, R2 and R3 and these are used to store the intermediate results that are required by the same ACS unit in a consecutive clock cycles. The second register level comprises Reg bank1 and Reg bank2 which are used to temporarily store the LUT-LOG-BCJR variables.

The number of operations to perform the max* is also reduced if we use constant log BCJR algorithm. The proposed architecture is shown in the figure 4.5. The proposed decoder processed two blocks at a time. So each decoder calculates soft output in forward path and backward path. The first step is to compute the branch metric value the branch metric value. The constant-log-MAP algorithm, approximates the Jacobi logarithm and the max* is calculated as

$$\text{Max}^*(p,q) = \begin{cases} 0 & \text{if } |y-x| > T; \ C \text{ if } |y-x|\leq T \end{cases}$$

(5)

Where $T$ is the threshold value and $C$ is the constant value. This algorithm is equivalent to the log-MAP algorithm with the correction function implemented by a 2-element look-up table. Therefore this algorithm reduced the hardware complexity. The value of $C$ is .5 value of $T$ is 1.5. These values are chosen to the best value for the calculation.
4.5 ACS UNIT

The ACS[12] (Add Compare Select) unit consists of two adders and compare and select unit consists max* unit. The adder unit calculates the sum of forward, branch and reverse state metric for zero transition branch and one transition branch.

R1 and R2 is loaded with the values of the adders. It takes input from Registers R1 and R2. Then given the result the R3.

This unit help to do addition subtraction and the max* calculation. If addition is to performed then opcode O is 00000 and for subtraction the opcode O is 10000

**Operation in the max* calculation**

**OPERATION 1:** O = 10110, In this clock cycle max* operation is calculated.

x~ and y~ are loaded from the register R1 and R2 and result is store in R3. Result C0 determine the max(R1,R2)

**OPERATION 2:** O = 11001, x~ is loaded with the value T |R1-R2| is loaded in the y~ .C1 determine the outcome of the test |R1-R2| > T

**OPERATION 3:** O = 00000, x~ is provided with maximum of R1 and R2. y~ is loaded with constant T. If C1 = 0, Max(R1,R2) is added with C. If C1=1 Max(R1,R2) is added with 0.

5 RESULT ANALYSIS

In order to verify the area efficiency and power efficiency, we have captured our new design in verilog. For performance comparison we have also implemented the FPGA. Following section gives the performance results of these implementations.

5.1 Area analysis

<table>
<thead>
<tr>
<th>parameters</th>
<th>Constant-Log-BCJR</th>
<th>Lut-Log-BCJR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory used</td>
<td>186712 kilobytes</td>
<td>187224 kilobytes</td>
</tr>
<tr>
<td>Delay</td>
<td>7.235 ns</td>
<td>7.241 ns</td>
</tr>
<tr>
<td>No. of lut elements</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

From this table it is clear that the memory, delay of the proposed architecture is less than that of the LUT-LOG architecture.

5.2 Power analysis

<table>
<thead>
<tr>
<th></th>
<th>Constant-Log-BCJR</th>
<th>Lut-Log-BCJR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>88MW</td>
<td>142MW</td>
</tr>
</tbody>
</table>

From this table it is clear that the power consumption of the proposed architecture is less than that of the LUT-LOG-BCJR.

**CONCLUSION**

The study of turbo codes starts from the error correction codes in the information theory. The turbo codes are derived from the convolution codes. It is the parallel concatenation of the recursive systematic convolutional codes. Turbo codes are considered to be having high BER performances. The turbo encoding is less complex, but decoding is difficult. There are many algorithms for the decoding. From the available algorithms the Constant log BCJR algorithm has less complexity and also has good BER. In the design of low complexity turbo decoder using Constant log BCJR algorithm process two blocks of data at a time. Although the hardware requirement is high, the advantage is that the throughput is more due to processing of two blocks of data at a time. When the proposed architecture is compared with the LUT log architecture it is found that the area and the power consumption is less. Also it has better BER performance.

**ACKNOWLEDGMENT**

We would like our sincere thanks to the Management and staff, SNG College of engineering for providing the facilities.

**REFERENCES**


