128 Bit Modified MLDD To Reduce Power Utilization And Fault Detection

Authors
Nikhat firdose jahan¹, P. Gopi Krishan²
¹M.Tech student in VLSI, Ramachandra college of Engineering, Eluru.
²Assistant Professor, Ramachandra college of engineering, Eluru.
Email-Nikki.firdose1113@gmail.com,Gopikrishna.popuri@gmail.com

ABSTRACT
To reduce the error in memory when the system is trends to read and the utilisation of power should be minimal for these reasons a redundant cyclic codes were proposed with 128 bit MLDD i.e., Majority logic decoder and detector. By using this method the MLDD is not only detecting the error and suppressing this in the later phase of the decoder. Here the entire process is runs on the side of decoder. However, this is only possible on the reception or decoding phase in identifying large number of errors and fault conditions of the circuits selected.

Keywords: Error, power, redundant cyclic code, MLDD, Decode and fault conditions.

INTRODUCTION
Soft error occurs when a radiation event causes enough of a charge disturbance to reverse or flip the data state of a memory cell. The soft error is also often referred to as a Single Event Upset (SEU). If the radiation event is of a very high energy, more than a single bit maybe affected, creating Multi Bit Upset (MBU). For reliable communication, errors must be detected and corrected. Error detection is the way to find out that is a data is correct or incorrect. Reed-Muller is one of the methods of multiple error detection in blocks for digital communications signals [2], [10]. Soft error reliability is to employ Error Detection and Correction (EDAC) techniques or Error Correction Codes (ECC) is employed. Various error detection techniques are used to avoid the soft error. One of the methods is majority logic decoder which used to detect and correct the error in simple way. The drawback of this method is increase the average latency of the decoding process because it depends on the size of the code. Another method is syndrome fault detector which increase the power consumption because it is complex module. Majority Logic Decoder/Detector (MLDD) is used for avoiding those drawbacks of existing methods [11].

MLDD is a method to decode the repetition codes. Repetition code is one of the most basic error-correcting codes. The idea of the repetition code is to just repeat the message several times. The hope is that the channel corrupts only a minority of
these repetitions. This way the receiver will notice that a transmission error occurred. Since the received data stream is not the repetition of a single message and moreover the receiver can recover the original message by looking at the received message in the data stream that occurs most often [9].

One specific type of LDPC codes, namely the Difference-Set Cyclic Codes (DSCCs) is based on the construction of a perfect difference set. Cyclic codes are a class of linear block codes that have convenient algebraic structures for efficient error detection and correction. Thus, we can find generator matrix (G) and parity check matrix (H). The reason is that they can be easily implemented with externally cost effective. Electronic circuit and the output show that the properties of DSCC-LDPC enable efficient fault detection [8], [12].

Various error detection techniques are used to avoid the soft error. One of the methods is majority logic decoder which used to detect and correct the error in simple way. The drawback of this method is increase the average latency of the decoding process because it depends on the size of the code. Another method is syndrome fault detector which increase the power consumption because it is complex module [4]. Majority Logic Decoder/Detector (MLDD) is used for avoiding those drawbacks of existing methods. The error correction codes that meet the requirements of higher error correction capability and low decoding complexity, cyclic block codes have been identified as good candidates, due to their property of being Majority Logic Decoder (MLD). Majority logic decoding is a method to decode repetition codes. Repetition code is one of the most basic error-correcting codes [6].

![Figure 1: Schematic of MLDD method.](image)

**SYSTEM MODEL**

Initially the input is stored into the cyclic shift register and it shifted through all the taps. The intermediate values in each tap are given to the XOR matrix which is used to perform the checksum equations. The resulting sums are then forwarded to the majority gate for evaluating its correctness. If the number of 1’s received is greater than the number of 0’s which would mean that the current bit under decoding is wrong, so it move on the decoding process of t It is used to produce the accurate result of the MLDD. Otherwise, the bit under decoding would be correct and no extra operations would be needed on it. Decoding process involving the operation of the content of the registers is rotated and the above procedure is repeated and it stops intermittently in the third cycle. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all is “0,” the
code word is determined to be error-free and forwarded directly to the output. If the error contains in any of the three cycles at least a “1,” it would continue the whole decoding process in order to eliminate the errors. Finally, the parity check sums should be zero if the code word has been correctly decoded. Finally the MLDD method is used to detect the five bit errors and correct four bit errors effectively. More than five bit errors it produces the output but it did not show the errors presented in the input. This type of error is called the silent data error. Drawback of this method is did not detecting the silent data error and it consuming the area of the majority gate. Overall operation of the MLDD is illustrated in figure 1.

Figure 2: MLDD Algorithm.

RELATED WORK

Figure 3: RTL SCHEMATIC of the system
RESULTS

Table 1: Device Utilization Report

<table>
<thead>
<tr>
<th>Device Utilization Summary (estimated values)</th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Utilization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Slices</td>
<td>157</td>
<td>24576</td>
<td>0%</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>231</td>
<td>49152</td>
<td>0%</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>238</td>
<td>49152</td>
<td>0%</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>150</td>
<td>640</td>
<td>23%</td>
</tr>
<tr>
<td>Number of GCLKs</td>
<td>1</td>
<td>32</td>
<td>3%</td>
</tr>
</tbody>
</table>

Figure 4: Report when there is no Error.

Figure 5: Report after 73 bits if error occurs and resolved in the very next bit.

Table 2: Timing Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Check</th>
<th>Worst Case Slack</th>
<th>Best Case Achievable</th>
<th>Timing Errors</th>
<th>Timing Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoTimespec constraint for clock net</td>
<td>SETUP HOLD</td>
<td>0.454ns</td>
<td>5.274ns</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6: Maximum power consumed by the circuit.
Figure 7: Minimum Power consumed by the circuit.

<table>
<thead>
<tr>
<th>Ambient Temperature</th>
<th>Clock Power</th>
<th>IOs Power</th>
<th>Leakage Power</th>
<th>Total Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>20°C</td>
<td>0.021</td>
<td>0.005</td>
<td>0.549</td>
<td>0.576</td>
</tr>
<tr>
<td>30°C</td>
<td>0.021</td>
<td>0.005</td>
<td>0.604</td>
<td>0.630</td>
</tr>
<tr>
<td>40°C</td>
<td>0.021</td>
<td>0.005</td>
<td>0.668</td>
<td>0.695</td>
</tr>
<tr>
<td>50°C</td>
<td>0.021</td>
<td>0.005</td>
<td>0.744</td>
<td>0.770</td>
</tr>
<tr>
<td>75°C</td>
<td>0.021</td>
<td>0.005</td>
<td>1.000</td>
<td>1.027</td>
</tr>
</tbody>
</table>

CONCLUSION

For the plain MLD, the memory read access delay is directly dependent on the code size (in this case a code with length 128 needs 128 cycles etc.) Then, for I/O two extra cycles needed. On the other hand, for proposed MLDD the memory read access delay is only dependent on the word error rate (WER). The proposed design just requires three cycles to detect any error (plus two of I/O). If an error is detected, all of the techniques need to run the whole decoding process.

REFERENCES

[6] Shih-Fu Liu, Pedro Reviriego, Member, IEEE, and Juan Antonio Maestro, Member, IEEE, Jan 2012, “Efficient Majority Logic Fault Detection With Difference-Set Codes for Memory


