An Efficient Single and Double-Adjacent Error Correcting Parallel Decoder for the (24, 12) Extended Golay Code

Palla Vasavi Devi[1], B.Venkateswramma[2]
M.Tech. PG Scholar, Gouthami Institute of Technology & Management for Women, Proddatur
Assistant Professor, Gouthami Institute of Technology & Management for Women, Proddatur

***Abstract*- Memories that operate in harsh environments, like for example space, suffer a significant number of errors. The error correction codes (ECCs) are routinely used to ensure that those errors do not cause data corruption. However, ECCs introduce overheads both in terms of memory bits and decoding time that limit speed. In particular, this is an issue for applications that require strong error correction capabilities. A number of recent works have proposed advanced ECCs, such as orthogonal Latin squares or difference set codes that can be decoded with relatively low delay. The price paid for the low decoding time is that in most cases, the codes are not optimal in terms of memory overhead and require more parity check bits. On the other hand, codes like the (24, 12) Golay code that minimize the number of parity check bits have a more complex decoding. A compromise solution has been recently explored for Bose–Chaudhuri–Hocquenghem codes. The idea is to implement a fast parallel decoder to correct the most common error patterns(single and double adjacent) and use a slower serial decoder for the rest of the patterns. In this brief, it is shown that the same scheme can be efficiently implemented for the (24, 12) Golay code. In this case, the properties of the Golay code can be exploited to implement a parallel decoder that corrects single- and double-adjacent errors that is faster and simpler than a single-error correction decoder. The evaluation results using a 65-nm library show significant reductions in area, power, and delay compared with the traditional decoder that can correct single and double-adjacent errors. In addition, the proposed decoder is also able to correct some triple-adjacent errors, thus covering the most common error patterns. The proposed architecture of this paper analysis is the logic size and area using Xilinx 14.3.**

***Index Terms*-Software quality assurance, Software engineering.**

I. INTRODUCTION

The Harsh environments, like space, are a challenge for electronic circuits in general and for memories in particular. For example, radiation causes several types of errors that can disrupt the circuit functionality. One common error for SRAM memories is soft errors that change the value of one or more memory cells. To avoid corruption in the data stored in the memory, error correction codes (ECCs) are commonly used. ECCs adds parity check bits to each memory word to detect and correct errors. This requires an encoder to compute those bits when writing to the memory and a decoder to detect and correct errors when reading from the memory. These elements increase the memory area and the power consumption, and can also reduce the access speed. These overheads increase with the error correction capability of the ECC. Traditionally, codes that can correct a single bit error per word have been used. In particular, single error correction–double error detection (SEC–DED) codes that can also detect double errors are commonly used. In recent years, the number of errors that affect more than one memory cell has increased significantly. This is due to the scaling of the memory cells and is projected to grow further. These errors, known as multiple cells upsets (MCUs), pose a challenge for SEC–DED codes. One solution to ensure that the MCU errors can be corrected is to interleave the bits of different logical words so that an MCU affects one bit per word. This is based on the observation that the cells affected by an MCU are physically close. Interleaving, however, has a cost as it complicates the memory design. In some space applications, there is an additional issue as the number of errors is high, and SEC–DED codes may not be sufficient when errors accumulate over time. These issues have led to an increased interest

on the use of more advanced ECCs to protect SRAM memories. As MCUs affect cells that are close together, a number of codes that can correct double-adjacent or triple-adjacent errors have been recently proposed. These codes, in many cases, do not require additional parity check bits and in the rest require only one or two additional bits. The decoding complexity increases but in many cases can still be implemented with limited impact on the memory speed. These codes are useful for applications in which the error rate is low; however, when the error rate is large, codes that can correct errors on multiple independent bits are needed. Research for multibit ECCs has focused on reducing the decoding latency as in many cases; the traditional decoders are serial and require several clock cycles. To some extent this can be done for some traditional ECCs by using a parallel syndrome decoder butthe decoder complexity explodes as the error correction capability or the word size increases. Another approach is to use codes that can be decoded with low delay, such as orthogonal Latin squares (OLSs) or difference set (DS) codes. In the case of OLS codes, the main issue is that they are not optimal in terms of the number of parity check bits and thus require more memory overhead. The DS codes are more competitive in terms of parity check bits but are still not optimal for some word lengths. For example, the (21, 10) DS code can correct 2-bit errors while a code with a similar block size and code rate, and the (24, 12) extended Golay code can correct 3-bit errors. However, the Golay code requires a more complex decoder that needs several clock cycles. Namba et al. Have proposed a compromise solution for Bose–Chaudhuri–Hocquenghem codes. The idea is that the most common error patterns are decoded in parallel and the rest serially. In particular, single and double-adjacent errors are corrected in a single clock cycle. This means that the most memory accesses can be completed in a single clock cycle, and only a small percentage of the words in error require a full serial decoding. This can enable the use of traditional ECCs that do not support fast parallel decoding to protect SRAM memories In this brief, the use of the scheme in is considered for the (24,12) Golay code. In more detail, an efficient parallel decoder capable of correcting the single and double-adjacent errors is presented. The decoder exploits the properties of the Golay code to reduce the implementation cost. This result in a decoder that is simpler than a traditional SEC decoder but that can also correct all double-adjacent errors and some triple-adjacent errors. The proposed decoder has been implemented in hardware description language and mapped to a 65- nm technology to show its benefits. The main contribution of this brief is to enable a fast and efficient parallel correction of the single and double-adjacent errors in the (24, 12) Golay code.

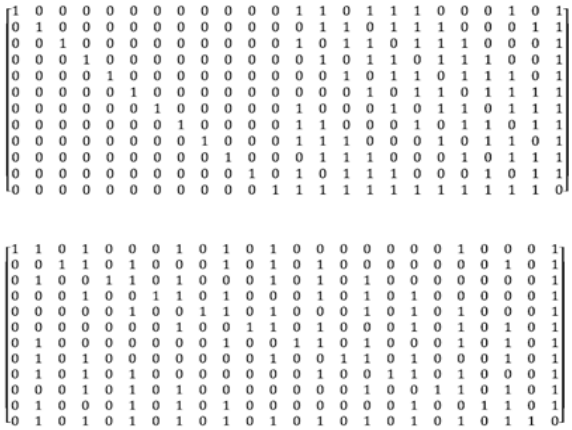


Fig. *Parity check matrix of the (24, 12) Golay Code with the proposed bit placement*

II. PRELIMINARIES

The ECCs add parity check bits to each memory word to detect and correct errors. This requires an encoder to compute those bits when writing to the memory and a decoder to detect and correct errors when reading from the memory. These elements increase the memory area and the power consumption, and can also reduce the access speed. These overheads increase with the error correction capability of the ECC. Traditionally, codes that can correct a single bit error

per word have been used. In particular, single error correction–double error detection (SEC–DED) codes that can also detect double errors are commonly used.

The use of the scheme is considered for the(24,12) Golay code. In more detail, an efficient parallel decoder capable of correcting the single and double-adjacent errors is presented.The decoder exploits the properties of the Golay code to reduce the implementation cost. This results in a decoder that is simpler than a traditional SEC decoder but that can also correct all double-adjacent errors and some triple-adjacent errors. The proposed decoder has been

implemented in hardware description language and mapped to a 65-nm technology to show its benefits. The main contribution of this brief is to enable a fast and efficient parallel correction of the single and double-adjacent errors in the (24,12) Golay code.

ERROR CORRECTION AND DETECTION:

In information theory and coding theory with applications in computer science and telecommunication, error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during

Transmission from the source to a receiver. Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data in many cases. The general definitions of the terms are as follows:

•Error detection is the detection of errors caused by noise or other impairments during TRANSMISSION from the transmitter to the receiver.

•Error correction is the detection of errors and reconstruction of the original, error-free data

OVERVIEW:

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted. Error-detection and correction schemes can be either systematic or non-systematic: In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of check bits (or parity data), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, an error has occurred at some point during the transmission. In a system that uses a non-systematic code, the original message is transformed into an encoded message that has at least as many bits as the original message. Good error control performance requires the scheme to be selected based on the characteristics of the communication channel. Common channel models include memory-less models where errors occur randomly and with a certain probability, and dynamic models where errors occur primarily in bursts. Consequently, error-detecting and correcting codes can be generally distinguished between random-error-detecting/correcting and burst-error-detecting/correcting. Some codes can also be suitable for a mixture of random errors and burst errors. If the channel capacity cannot be determined, or is highly variable, an error-detection scheme may be combined with a system for retransmissions of erroneous data. This is known as automatic repeat request (ARQ), and is most notably used in the Internet. An alternate approach for error control is hybrid automatic repeat request (HARQ), which is a combination of ARQ and error-correction coding.

IMPLEMENTATION:

Error correction may generally be realized in two different ways:

Automatic repeat request (ARQ) (sometimes also referred to as backward error correction): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data. Every block of data received is checked using the error detection code used, and if the check fails, retransmission of the data is requested – this may be done repeatedly, until the data can be verified.

Forward error correction (FEC): The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data.

ARQ and FEC may be combined, such that minor errors are corrected without retransmission, and major errors are corrected via a request for retransmission: this is called hybrid automatic repeat-request (HARQ)

III. IMPORTANCE OF HDLS

HDLs have many advantages compared to traditional schematic-based design.

•Design can be described at a very abstract level by us of HDLs.Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate level netlist,using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.

•By describing designs in HDLs,functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized.

•Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs.

•HDL-based designs are here to stay. With rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools,HDLs are now the dominant method for large digital designs. No digital circuit designer can afford to ignore HDL based design.

Verilog HDL has evolved as a standard hardware description language. Verilog HDL offers many useful features

•Verilog HDL is a general-purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to the C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.

•Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer needs to learn only one language for stimulus and hierarchical design.

•Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers.

•All fabrication vendors provide Verilog HDL libraries for post logic synthesis simulation. Thus, designing a chip in Verilog HDL allows the widest choice of vendors.

•The Programming Language Interface (PLI) is a powerful feature that allows the user to write custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their needs with the PLI.

•The speed and complexity of digital circuits have increased rapidly. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With designer assistance, EDA tools have become sophisticated enough to achieve a close-to-optimum implementation.

•The most popular trend currently is to design in HDL at an RTL level, because logic synthesis tools can create gate-level net lists from RTL level design. Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design.

•However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies.

•Formal verification and assertion checking techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate-level net lists. However, the need to describe a design in Verilog HDL will not go away. Assertion checkers allow checking to be embedded in the RTL code. This is a convenient way to do checking in the most important parts of a design.

•New verification languages have also gained rapid acceptance. These languages combine the parallelism

and hardware constructs from HDLs with the object oriented nature of C++. These languages also provide support for automatic stimulus creation, checking, and coverage. However, these languages do not replace Verilog HDL. They simply boost the productivity of the verification process. Verilog HDL is still needed to describe the design.

•For very high-speed and timing-critical circuits like microprocessors, the gate-level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high-speed designs because designers need to squeeze the last bit of timing out of circuits, and EDA tools sometimes prove to be insufficient to achieve the desired results.

•Another technique that is used for system-level design is a mixed bottom-up methodology where the designers use either existing Verilog HDL modules, basic building blocks, or vendor-supplied core blocks to quickly bring up their system simulation. This is done to reduce development costs and compress design schedules. For example, consider a system that has a CPU, graphics chip, I/O chip, and a system bus.

•The CPU designers would build the next-generation CPU themselves at an RTL level, but they would use behavioral models for the graphics chip and the I/O chip and would buy a vendor-supplied model for the system bus. Thus, the system-level simulation for the CPU could be up and running very quickly and long before the RTL descriptions for the graphics chip and the I/O chip are completed.

TYPICAL DESIGN FLOW:

A typical design flow for designing VLSI-IC circuits show the level of design representation shaded blocks show processes in the design flow. The design flow used by designers who use HDLs. In any design, specifications are written first. Specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. At this point, the architects do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, and compliance to standards, and other high-level issues. Behavioral descriptions are often written with HDLs. The behavioral description is manually converted to an RTL description in an HDL.

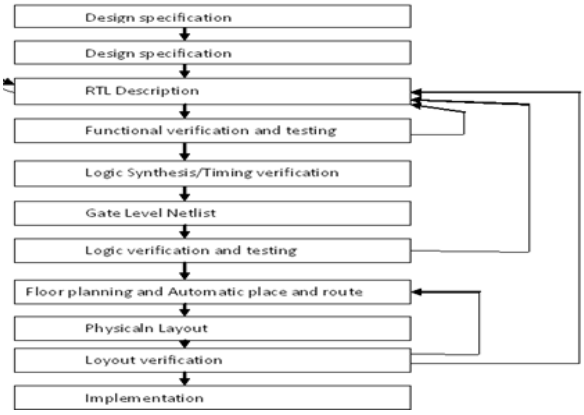


Fig. *Typical Design Flow*

Logic synthesis tools convert the RTL description to a gate-level net list. A gate-level net list is a description of the circuit in terms of gates and connections between them. Logic synthesis tools ensure that the gate-level net list meets timing, area, and power specifications. The gate-level net list is input to an Automatic Place and Route tool, which creates a layout. The layout is verified and then fabricated on a chip.

Thus, most digital design activity is concentrated on manually optimizing the RTL description of the circuit. After the RTL description is frozen, EDA tools are available to assist the designer in further processes. Designing at the RTL level has shrunk the design cycle times from years to a few months. It is also possible to do many design iterations in a short period of time. Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit. As these tools mature, digital circuit design will become similar to high-level computer programming. Designers will simply implement the algorithm in an HDL at a very abstract level. EDA tools will help the designer convert the behavioral description to a final IC chip.

It is important to note that, although EDA tools are available to automate the processes and cut design cycle times, the designer is still the person who controls how the tool will perform. EDA tools are also susceptible to the "GIGO : Garbage In Garbage Out" phenomenon. If used improperly, EDA tools will lead to inefficient designs. Thus, the designer still needs to understand the nuances of design

methodologies, using EDA tools to obtain an optimized design.

IV. (24, 12) EXTENDED GOLAY CODE

The (24, 12) extended Golay code is obtained by adding an overall parity check bit to the (23, 12) Golay code. This code is a perfect code with a minimum distance of seven and has been widely studied. The extended code has a minimum distance of eight, and therefore can correct 3-bit errors and detect 4-bit errors. It has been used in many applications including space missions that require strong error correction capabilities. The decoding of the Golay code is done in a series of steps, and requires several clock cycles. For example, 27 clock cycles are needed in the implementation presented in. This, as discussed before, is not suitable for SRAM protection. To the best of our knowledge, no SEC-DAEC parallel decoder optimized for the Golay code has been proposed in the literature. The parity check matrix of the (24, 12) Golay code is shown in Fig. 1. The 12 first bits correspond to the parity check bits and last 12 to the data bits. A single-error correcting parallel decoder can be implemented by computing the syndrome and comparing in parallel with the 12 data bit and the 12 check bit columns. When there is a match that bit is corrected. The requirement for SEC is that the columns must be different. Therefore, it would seem possible to use a subset of the parity bits to decode single errors. However, since the code can correct three errors, we need to ensure that the single-error parallel decoder does not introduce erroneous corrections in the presence of multiple bit errors. For example, if we use an SEC-DAEC code with a minimum distance of four, a triple error can cause a mis correction in the SEC-DAEC decoding phase. A 4-bit error may not be even detected by the SEC-DAEC decoder. Therefore, the full syndrome is used for comparisons in all the cases to ensure that triple errors do not trigger miscorrections and 4-bit errors are detected.

Proposed sec-daec parallel decoder:

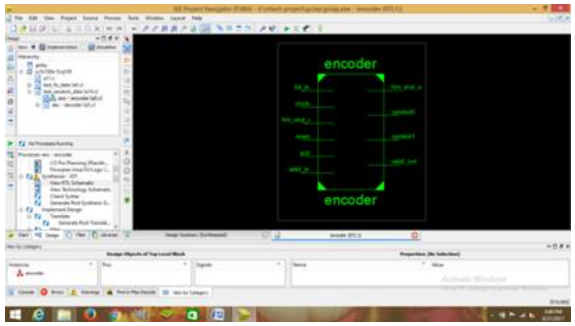
The existing SEC-DAEC decoders are similar to SEC decoders but they need to check also the syndrome values that correspond double adjacent errors. This requires roughly doubling the number of comparisons. Then, the correction of each bit is triggered by three syndrome values (the single bit and the two double adjacent). This results in a decoder that is significantly more complex than a simple SEC decoder. The proposed parallel decoder as discussed before has the objective of correcting single and double-adjacent bit errors. The first step is to place the bits in the memory such that data and parity bits are interleaved, as shown in Fig. 2. This interleaving has no impact on memory performance, as it is a simple remapping of the bits when they are read from or written to the memory. Let us now consider the syndrome values for an error on the second bit (first data bit), a double adjacent on bits one and two, a double adjacent on bits two and three, and a triple adjacent on bits one, two, and three. In all those cases, bit two should be corrected. The syndrome values for those error patterns are shown in Fig. 3. The interesting observation is that the first two rows are the only ones that change from one pattern to another and that the values cover the four possible combinations of the first two bits. This means that the decoding can be done by simply comparing the remaining ten bits with the last ten bits of the syndrome. If they match, then the second bit (first data bit) has to be corrected. It can be observed that the same reasoning applies to the rest of the data bits, except the last one. For the last bit, there are only two values to check (single and double adjacent with bit 23). In this case, it is easy to see that this can be done by checking the first 11 bits only. The previous discussion shows how parallel decoding can be efficiently implemented. In fact, the proposed parallel decoder will be simpler than an SEC decoder. Table I summarizes the comparators needed for each of the different decoders. A comparator is needed for each syndrome value that triggers a correction. For an SEC code, this is simply 24 while for a traditional SEC-DAEC code is 47. In the case of the proposed decoder, 12 comparators cover both single bit errors on the data bits and double adjacent bit errors, and another 12 are needed to cover single errors on the check bits giving a total of 24. It can be observed that the proposed decoder needs less comparator and also less bits in some of them. Both factors help to reduce the decoder complexity. In Section IV, the benefits will be evaluated for a design mapped to a 65-nm technology. The proposed parallel decoder also has to detect errors that it cannot correct. In those cases, the serial decoder must be used to correct the error. The logic needed to detect those errors is simply a check for a no zero syndrome and a check that none of the comparators has detected a match. The first part can be implemented with a 12-input OR gate and the second with another 24-input OR gate. It should be noted that the same idea can be partly applied to other triple ECCs even if the number of parity check and data bits is not the same. In more detail, when there are more data bits, the first data bits can also be interleaved with parity bits and decoded with the proposed scheme, while for the rest, a traditional SEC-DAEC decoding can be used. The application of the proposed scheme to other codes is left for future

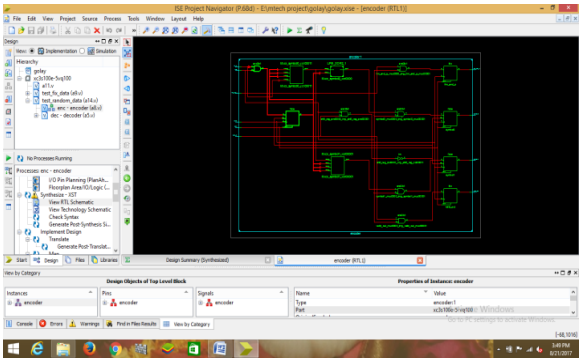
work.

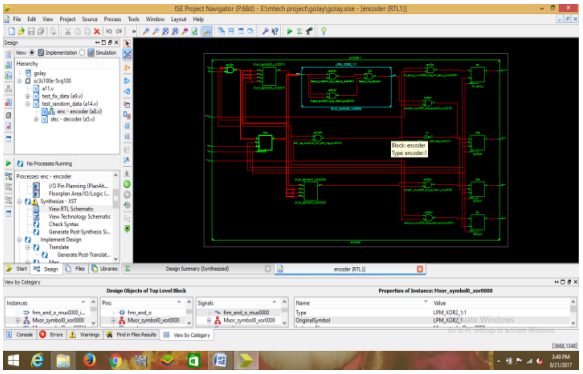
1	0	1	0
0	0	1	1
1	1	1	1
0	0	0	0
0	0	0	0
0	0	0	0
1	1	0	1
1	1	1	1
1	1	1	1
0	0	0	0
1	1	1	1
1	1	1	1

Fig. *Example syndrome values for errors that effects first data bit in the proposed bit placement*

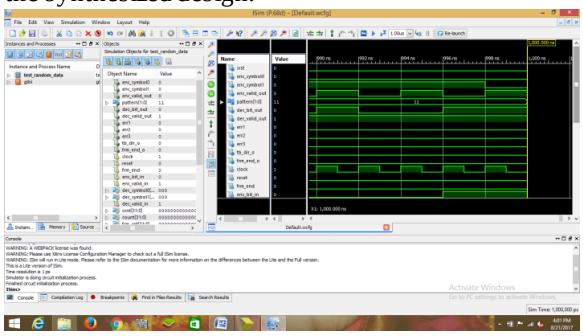
V.SIMULATION TOOLS

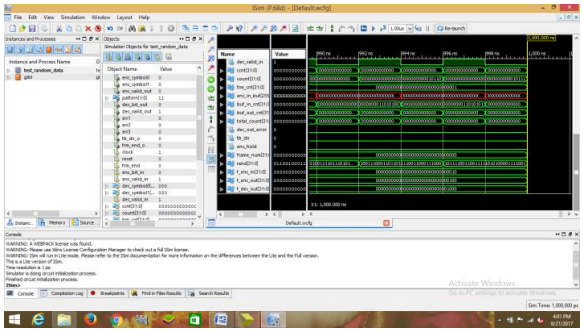


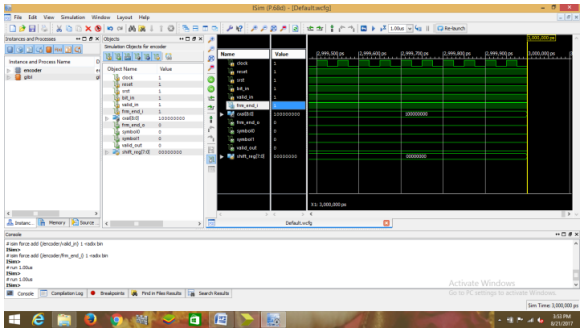




The below figure shows the behavioral simulation of the synthesized design.







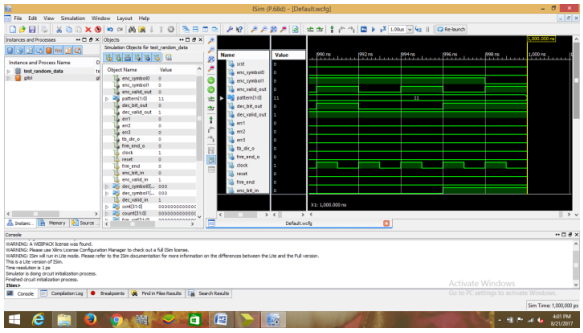


Fig: Simulation Result

VI. CONCLUSION

In this brief, a single and double-adjacent error correcting parallel decoder for the (24, 12) extended Golay code has been proposed. The decoder uses the properties of the code to achieve an efficient implementation. In fact, the proposed decoder is not

only much simpler than a traditional SEC-DAEC decoder, but also simpler than a standard SEC decoder for the Golay code. To evaluate the benefits of the new decoder, it has been implemented in HDL and mapped to a 65-nm library. The results confirm that significant reductions in area, delay, and power consumption can be obtained compared with the traditional SEC-DAEC decoder. The new SEC-DAEC parallel decoder can be used in conjunction with a serial decoder so that the most common error patterns are corrected in one clock cycle

REFERENCES

[1] R. D. Schrimpf and D. M. Fleetwood, Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices. Singapore:World Scientific, 2004.

[2] R. C. Baumann, "Soft errors in advanced computer systems," IEEE Des. Test. Compute., vol. 22, no. 3, pp. 258–266, May/Jun. 2005.

[3] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," IBM J. Res. Develop., vol. 28, no. 2, pp. 124–134, Mar. 1984.

[4] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," IBM J. Res. Develop., vol. 14, no. 4, pp. 301–395, Jul. 1970.

[5] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule," IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[6] P. Reviriego, J. A. Maestro, S. Baeg, S. Wen, and R. Wong, "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," IEEE Trans. Nucl. Sci., vol. 57, no. 4, pp. 2124–2128, Aug. 2010.

[7] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's," IEEE ElectronDevice Lett., vol. 21, no. 6, pp. 310–312, Jun. 2000.

[8] A. Neale and M. Sachdev, "A new SEC-DED error correction code subclass for adjacent MBU tolerance in embedded memory," IEEETrans. Device Mater. Rel., vol. 13, no. 1, pp. 223–230, Mar. 2013.

[9] M. A. Bajura et al., "Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs," IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 935–945, Aug. 2007.

[10] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in Proc. IEEEVLSI Test Symp., May 2007, pp. 349–354.

[11] Z. Ming, X. L. Yi, and L. H. Wei, "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory," in Proc. IEEE/IFIP 20th Int.Conf. VLSI Syst.-Chip, Oct. 2011, pp. 254–259.

[12] L.-J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, and J. A. Maestro, "MCU tolerance in SRAMs through low-redundancy triple adjacent error correction," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., to be published.

[13] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100 nm technologies," in Proc. IEEE ICECS, Aug./Sep. 2008, pp. 586–589.

[14] M. Y. Hsiao, D. C. Bossen, and R. T. Chien, "Orthogonal Latin square codes," IBM J. Res. Develop., vol. 14, no. 4, pp. 390–394, Jul. 1970.

[15] S.-F. Liu, P. Reviriego, and J. A. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications," IEEETrans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 1, pp. 148–156, Jan. 2012.

[16] M. J. E. Golay, "Notes on digital coding," Proc. IEEE, vol. 37, p. 657, Jun. 1949.

[17] S. Sarangi and S. Banerjee, "Efficient hardware implementation of encoder and decoder for Golay code," IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst., to be published.

[18] K. Namba, S. Pontarelli, M. Ottavi, and F. Lombardi, "A single-bit and double-adjacent error correcting parallel decoder for multiple-bit error correcting BCH codes," IEEE Trans. Device Mater. Rel., vol. 14, no. 2, pp. 664–671, Jun. 2014.

[19] S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[20] E. R. Berlekamp, "Decoding the Golay code," Jet Propulsion Lab., Pasadena, CA, USA, Tech. Rep. 32-1526, 1971, pp. 81–84.