Journal of Engineering Sciences
Vol 13, Issue 06, June/2022
ISSN NO:0377-9254

LOCALIZATION AND DEBUGGING OF MULTIPLE FAULTS IN ARITHMETIC CIRCUITS

Mr.P.Bose Babu,M-Tech(Ph.D)
Professor Department of ECE
Andhra Loyola Institute of Engineering and Technology,Vijayawada, India.
E-mail:pbosebabu@aliet.ac.in

Mule.Madhava Reddy
Student,DepartmentofECE, Andhra Loyola Institute of Engineering and Technology,Vijayawada, India.
E-mail:mulemadhava724@gmail.com

Kosuri Charan Naga Sai
Student, Department of ECE, Andhra Loyola Institute of Engineering and Technology, Vijayawada, India.
E-mail:charannagasai2001@gmail.com

Puppala Dunesh Siva Kumar
Student, Department of ECE, Andhra LoyolaInstitute of Engineering and Technology, Vijayawada, India.
E-mail:duneshpuppala@gmail.com

***Abstract*—Optimised and custom arithmetic circuits are widely used in embedded systems such as multimedia applications,cryptography systems,signal processing and console games.Debugging of arithmetic circuits is a challenge due to increasing complexity coupled with non standard implementations.However,bug localization remains a major issue.simulation based validation using random or constrained-random tests are not effective for complex arithmetic circuits due to bit-blasting.In this model it will present an automated test generation and bug localization technique for debugging arithmetic circuits**

Finally our model is fully automated.In other words this model is capable of producing a correct implementation of arithmetic circuiots with out any manual interaction.Our experimental results demonstrate that the proposed approach can br used for automated debugging of large and complex arithmetic circuits

Keywords—Arithmetic circuits,Multiplier,Bugs,Independen tbug,Dependent bug,Hybrid bug,Error Localization,Error Detection and Error Correction.

I. INTRODUCTION

A bug is just an error that has to be corrected in localization words. All problems (errors) found during localization testing are recorded in a database that keeps track of these entries. Then, a tester assigns each bug's resolution to the proper party, based on the software's knowledge requirements, linguistic ability, and source of the mistake. For instance, developers must fix various internationalisation problems. Most problems in source code or its design are the result of human error. When a programme has numerous defects that interfere with its functionality and produce unintended outcomes, it is said to be buggy.

Software bugs can result from misunderstandings and errors occurring when deriving requirements from users, planning a program's design, developing its source code, and interacting with people, hardware, and other programmes, including operating systems or libraries. Often, a programme is said to be buggy if it has numerous, or significant, defects. Errors caused by bugs may have repercussions. The effects of bugs can range from subtle, like unexpected text formatting, to more overt, such crashing a programme, locking up the computer, or breaking hardware. Other flaws fall under the category of security flaws and could, for instance, allow a hostile user to evade access controls and get unauthorised rights.

II. LITERATURE SURVEY

In [1] In order to rectify all provided counterexamples, this approach finds logic gates that need to be swapped out for alternative logic functions
In [2] The introduction of bug patterns lengthens the time that SMT solvers take to run. We provide a more effective approach that uses programme slicing and constraints created from the counterexamples to decrease the running time while also reducing the quantity of the design description that needs to be translated to an SMT formula and the number of bug candidates
In [3] The introduction of bug patterns lengthens the time that SMT solvers take to run. We provide a more effective approach that uses programme slicing and constraints created from the counterexamples to decrease the running time while also reducing the quantity of the design description that needs to be translated to an SMT formula and the number of bug candidates.
In [4] Bugs must be detected and repaired as early in the development process as is practical. This is the main rule for semiconductor design and verification. It's commonly believed that finding a bug at each new project stage increases the cost by 10.
In [5] The standard specifies how Verilog HDL's semantics are used, among other things, to specify level- and edge-sensitive logic. Additionally, it explains the language's grammar in terms of what must be supported and what must not be supported for interoperability
In [6] We provide BLUIR, a structured retrieval method that incorporates our insights, and demonstrate that BLUIR outperforms the current cutting-edge IR-based bug localization techniques
In [7] They developed Csmith, a tool for randomly generating test cases, to enhance the quality of C compilers, and they spent three years hunting for compiler flaws with it. Over the course of this time, we notified compiler

developers of more than 325 previously undiscovered problems.

III. PROPOSED ARCHITECTURE

The primary goal of the project is to detect and correct the faults in arithmetic circuits,we are using three types of bugs in our proposed model those are Independent bug,Dependent bug and Hybrid bud.To detect and correct these bugs,the algorithms used are Error Localization,Error Detection and error Correction.In this proposed model we used process called BIST(Built In Self Test).

A. Generating the Multiplier
B. Synthesizing
C. Bug Localization
D. Bug Correction

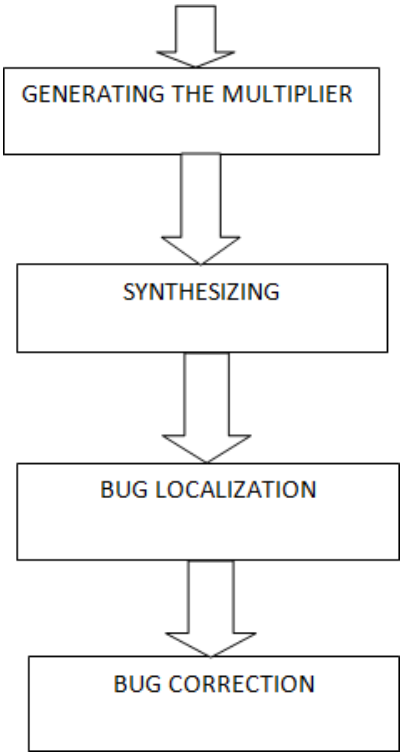

Figure1: Proposed Model

1.Multiplier

Many high performance systems depend on multipliers as essential parts. However, as speed and area are frequently at odds with one another, increasing speed primarily leads to larger areas. As a result, a wide range of multipliers with various area-speed constraints have been created using fully parallel architecture. These multipliers perform only averagely in terms of speed and area. No matter how big the multiplier is, the benefit of pipelining at the digit level is constant operation speed.

2. 2-Bit Multiplier

By multiplying each bit of the multiplicand by each bit of the multiplier to get the partial products, the circuit implements a two-bit by two-bit multiplier. The total product is then calculated by weighting and adding the partial products.

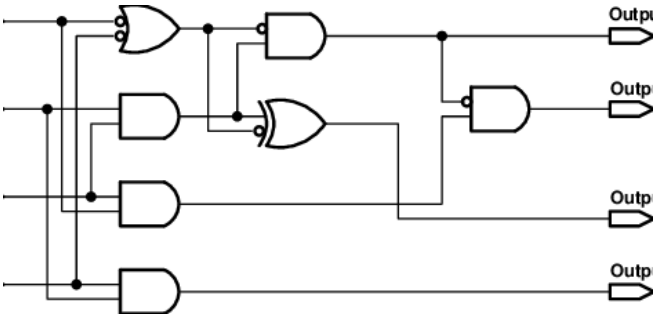

Figure 2 :2-Bit Multiplier

3 .4-bit Multiplier

The algorithm will finish in no more than 4 cycles for a 4-bit multiplication. The method is only lengthy multiplication. The lengthy multiplication of two 4-bit data to yield an 8-bit output is seen below.

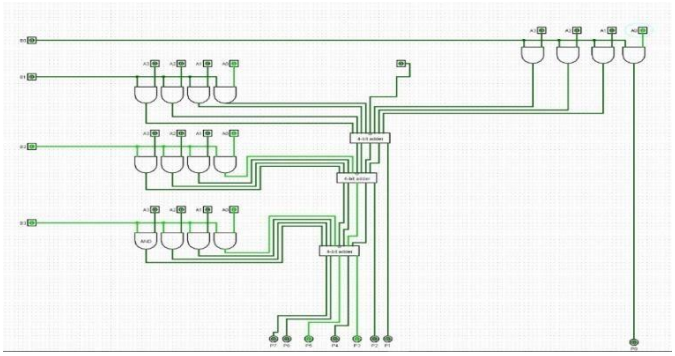

Figure 3 :4-Bit Multiplier

IV.Results

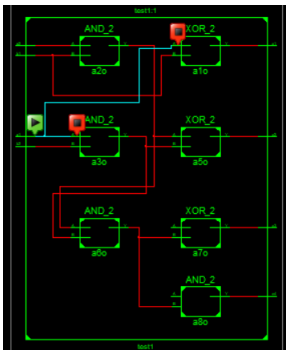

Figure 4 : Faulty 2-Bit Independent RTL Schematic Circuit

The above figure shows a schematic circuit with an independent bug present inside the circuit, where a1 a2 b1 and b2 are the inputs and o1 o2 o3 and o4 are the outputs of the circuit, where as o5,o6,o7 and o8 are the bug outputs. When outputs o1 and o5 are different, then an error e1 arises. But when the outputs o1 and¬ o5are same then there will not be any error. (Error free).

When outputs o2 and o6 are different, then an error e2 arises. But when the outputs o2 and o6¬ are same then there will not be any error. (Error free).

When outputs o3 and o7 are different, then an error e3 arises. But when the outputs o3 and o7¬ are same then there will not be any error. (Error free).

When outputs o4 and o8 are different, then an error e4 arises. But when the outputs o4 and o8¬ are same then there will not be any error. (Error free)

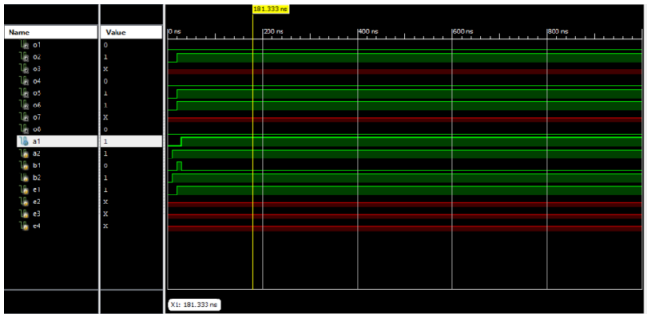

Figure 5 : Error Localization of 2-Bit Independent Circuit

In the above figure o1 and o5 are not equal so error is localized to correct the error we have to use the error correction logic using Xilinx tool.

Figure 6 : Error Correction of 2-Bit Independent Circuit

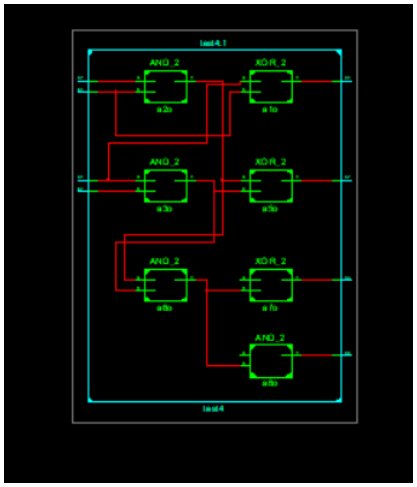

Figure 7 : Faulty 2-Bit Dependent RTL Schematic Circuit

Figure 8 : Error Localization of 2-Bit Dependent Circuit

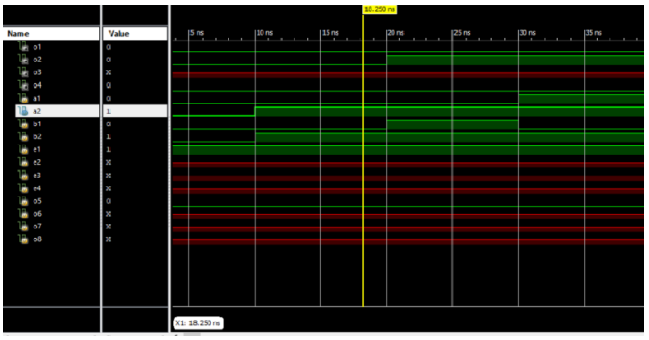

Figure 9 : Error Correction of 2-Bit Dependent Circuit

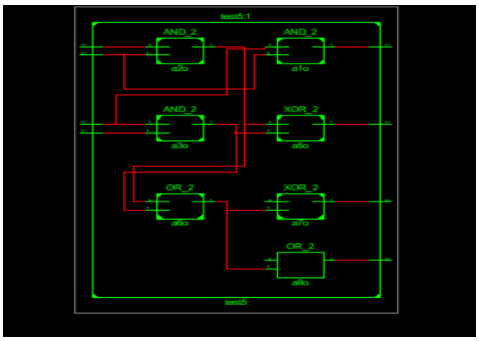

Figure 10 : Faulty 2-Bit Hybrid RTL Schematic Circuit

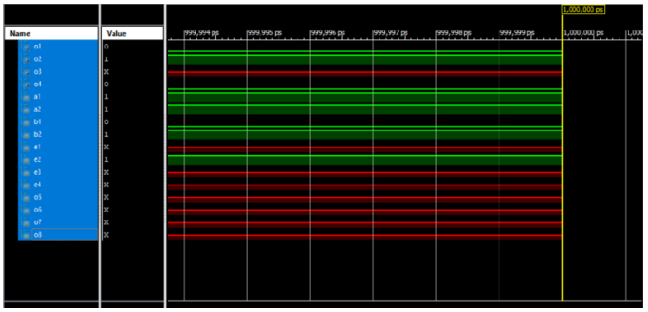

Figure 11 : Error Localization of 2-Bit Hybrid Circuit

Figure 12 : Error Correction of 2-Bit Hybrid Circuit

Journal of Engineering Sciences
Vol 13, Issue 06, June/2022
ISSN NO:0377-9254

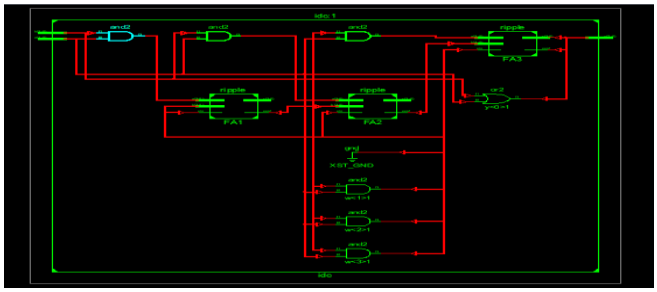

Figure 13 : Faulty 4-Bit Independent RTL Schematic Circuit

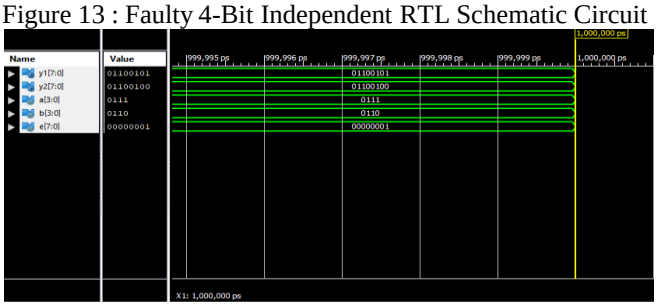

Figure 14 : Error Localization of 4-Bit Independent Circuit

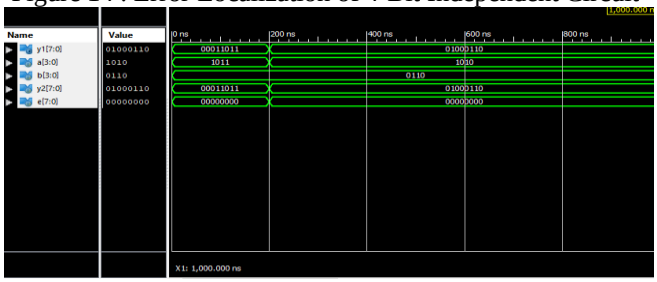

Figure 14 : Error Correction of 4-Bit Independent Circuit

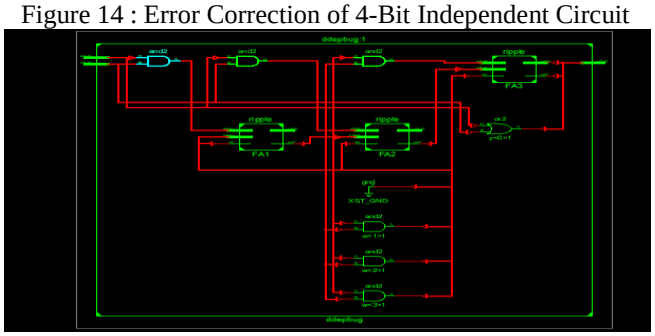

Figure 13 : Faulty 4-Bit Dependent RTL Schematic Circuit

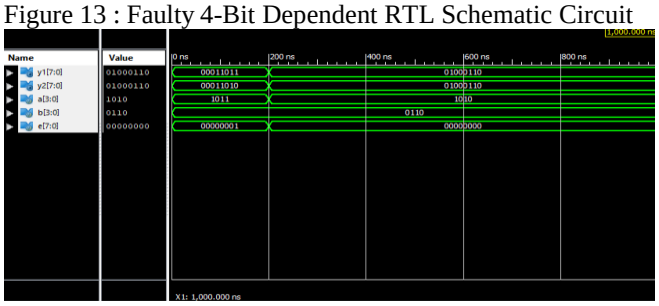

Figure 15 : Error Localization of 4-Bit Dependent Circuit

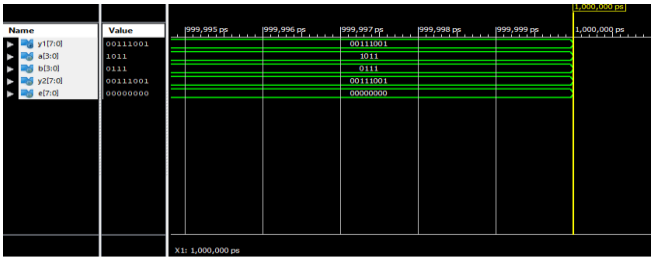

Figure 15 : Error Correction of 4-Bit Dependent Circuit

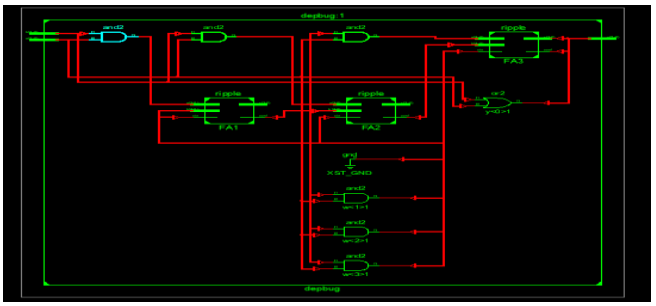

Figure 16 : Faulty 4-Bit Hybrid RTL Schematic Circuit

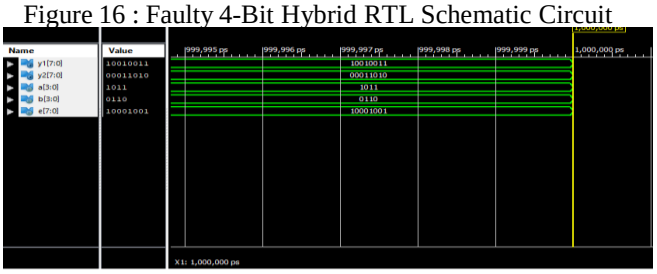

Figure 17 : Error Localization of 4-Bit Hybrid Circuit

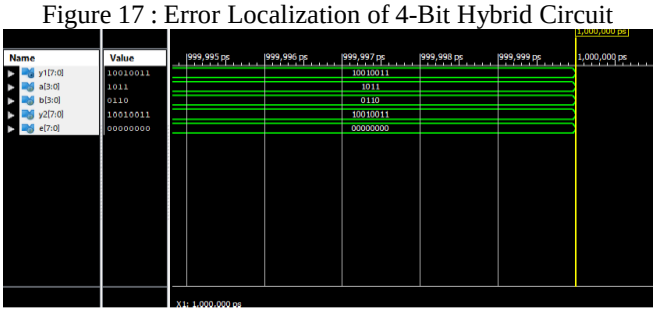

Figure 18 : Error Correction of 4-Bit Hybrid Circuit

Comparision of different parameters of 2-Bit multiplier and 4-Bit Multiplier

Test Cases	Number of LUT'S Used	Maximum Path Delay	CPU TIME	Total Time
Test-1 (localization)	3	9.063ns	5.94ns	11.75ns
Test-1(correction)	3	9.063ns	5.81ns	
Test-2 (localization)	3	9.109ns	5.89ns	9.34ns
Test-2(correction)	3	9.109ns	3.45ns	
Test-3 (localization)	3	9.063ns	3.44ns	6.87ns
Test-3(correction)	3	9.063ns	3.43ns	
Test-4 (localization)	3	9.063ns	3.47ns	7.00ns
Test-4(correction)	3	9.063ns	3.53ns	
Test-5 (localization)	3	9.109ns	3.63ns	7.13ns
Test-5(correction)	3	9.109ns	3.50ns	

Table 1 : comparision of different parameters for 2-Bit Multiplier

Test Cases	Number of LUT'S Used	Maximum Path Delay	CPU TIME	Total Time
Test-1 (localization)(Independent bug)	23	3.839ns	14.45ns	27.00ns
Test-1 (correction)(Independent bug)	23	3.839ns	12.55ns	
Test-2 (localization) (Dependent bug)	23	3.839ns	12.99ns	25.49ns
Test-2 (correction) (Dependent bug)	23	3.839ns	12.50ns	
Test-3 (localization) (Hybrid bug)	23	3.839ns	13.39ns	26.11ns
Test-3(correction)(Hybrid bug)	23	3.839ns	12.72ns	

Table 2 : comparision of different parameters for 4-Bit Multiplier

V. CONCLUSION

In the proposed model, the automated methodology for debugging arithmetic circuits. Our methodology consists of efficient , bug localization, and bug correction algorithms. We used the BIST produced by equivalence checking methods to generate directed tests that are guaranteed to activate the source of the bug when the bug is unknown. We used the generated tests to localize the source of the bug and find suspicious areas in the design. We also developed an efficient debugging algorithm that to locate and correct the bug without any manual intervention. We extended the proposed approach to automatically fix multiple bugs for 2-bit and 4-bit multiplier.

REFERENCES

[1] M. J. Ciesielski, C. Yu, W. Brown, D. Liu and A. Rossi, "Verification of gate-level arithmetic circuits by function extraction," in IEEE/ACM International Conference on Computer Design Automation(DAC), 2015, pp.

[2] A. Sayed-Ahmed, D. Große, U. Kuhne, M. Soeken, and R. Drech- ¨ sler, "Formal verification of integer multipliers by combining grobner basis with logic reduction," in ¨ Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016. IEEE, 2016, pp. 1048–1053..

[3] M. Chen and P. Mishra, "Functional test generation using efficient property clustering and learning techniques," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29, no. 3, pp. 396–404, 2010

[4] M. Chen, X. Qin, H. Koo, and P. Mishra, System-level Validation - high-level modeling and directed test generation techniques. Springer, 2012.

[5] J. Lv, P. Kalla and F. Enescu, "Efficient grbner basis reductions for formal verification of galois field multipliers," in Design Automation and Test in Europe Conference(DATE), 2012, pp. 899–904.

[6] F. Farahmandi and B. Alizadeh, "Grobner basis based formal verification of large arithmetic circuits using gaussian elimination and cone-based polynomial extraction," in Microprocessor and Microsystems - Embedded Hardware Design, 2015, pp. 83–96

[7] F. Farahmandi, P. Mishra, and S. Ray, "Exploiting transaction level models for observabilityaware post-silicon test generation," in Proceedings of the 2016 Conference on Design, Automation & Test in Europe. EDA Consortium, 2016, pp. 1477–1480

[8] D. Ritirc, A. Biere, and M. Kauers, "Improving and extending the algebraic approach for verifying gate-level multipliers," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018. IEEE, 2018, pp. 1556–1561.

[9] Y. Lyu, X. Qin, M. Chen, and P. Mishra, "Directed test generation for validation of cache coherence protocols," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[10] A. Ahmed, F. Farahmandi, and P. Mishra, "Directed test generation using concolic testing on rtl models," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018. IEEE, 2018, pp. 1538–1543