DETECTION OF MODULE INTEGRATION ERRORSIN HIERARCHICAL CIRCUIT DESIGNS

Nicholas Dematteis, Jesus Godinez, Gina Rhoads, and Maddu Karunaratne

Electrical & Computer Engineering, University of Pittsburgh, Johnstown, PA, USA

ABSTRACT

Large circuits are developed by integration of many smaller modules that have already been designed and verified for accurate functionality. The integration phase also requires verification that the signal connections between the integrated modules are correctly paired to produce the expected final design. The validation of the module connectivity may be performed either visually or via simulations as part of the verification step of the overall functionality which may not check every connection between modules .Inadvertently some module interconnections may be in error, and neither be discovered visually nor detected if stimulus does not cover the whole circuit. This research attempts to automate the validation of module interconnections in a select area or the entire circuit by creating stimulus for circuit simulations. First step in this work is to identify the types of errors that are likely to occur when individual modules are integrated to create the final circuit. The second step is to develop a software program to create test stimulus targeting the module interconnections so they can be simulated to produce circuit output values. If those values deviate from what is functionally expected, then module connection errors are present provided each module has been verified before the integration.

The research first classified the most probable alternative connections between modules into five categories based on the structure of the given hierarchical circuit. Secondly, algorithms were developed to enumerate possible alternatives to the interconnections between the modules in the circuit. These alternatives were considered as faults to be detected at the circuit boundary. Thirdly, tests were generated to detect such faults which yields external input data vectors to distinguish each potential alternative from the given circuit connections. Detection of a fault imply that the vector applied may be used for simulation of an equivalent behaviour or functional model of the circuit to obtain expected boundary values which would reveal whether the alternative is the correct connection or not. The test vectors were simulated against all the enumerated alternative connections (classified as faults) of the entire circuit to ensure each vector is productive in detecting a fault. The developed software was tested on a series of small circuits and appropriate benchmark circuits with great success. The generated test vectors distinguished (detected) 77% to 100% of all potential alternative interconnections between the modules in the circuits.

KEYWORDS

Design Error, Module Interconnections, Test Generation, Fault Model, Fault Simulation

1. INTRODUCTION

Verification is time consuming although it is an integral part of digital circuit design, in which a designer develops data stimuli (vectors) to apply to the circuit boundary in order to see how it produces results at the output terminals. This circuit boundary may be either external at the highest hierarchy level or at a lower module boundary. The stimulus is applied to the circuit via logical simulations and the designer compares the produced results against the expected values. The comparison may be integrated as part of the simulation environment. This process is repeated until the designer is satisfied that the circuit meets the high-level requirements of the system. While the simulation models of the circuit can automate the comparison process, it does not eliminate the effort in developing the test vectors to apply. To compound this issue, when multiple circuit modules are integrated together at multiple hierarchy levels, the interconnection and feedback paths may introduce some manual editing errors into the design which may escape detection if the test stimuli are not thorough enough.

Published research articles are available in very broad categories and most of those focus on finding and correcting errors in manufactured circuits. Some authors have put forth research into design errors in specific categories of circuits under various circumstances. Kang and Szygenda [1, 2] described many different design errors but did not cover hierarchical modules, boundary mismatches and resulting sequential loops, or tri state gates where three value logic is needed.Huang [3] analyzed stuck at faults in sequential circuits in an attempt to diagnose design errors in such circuits. Chang [4] did similar research but constrained the analysis to the circuit layout level only. Other research [5] used mathematical models to determine what went wrong in the designor to implement error correction schemes built into the circuit itself rather than detection via simulation.

In this research we developed a method and accompanying software that would detect common module interconnection mistakes for generic hierarchical circuit designs using external stimuli, with the goal of reducing the effort required for verification of the integrated circuit. We assume that each individual low-level module is accurate by design but their interconnections in making larger modules in a hierarchical design may introduce occasional port connection errors. Therefore, some or a few module connections in a given circuit design may not be correct and an alternate connection might have been the right design intention. This research work developed algorithms to identify and enumerate possible alternative connections, and software was developed to produce effective test stimuli to show different outcomes in the simulation values that depend on each distinct alternate connection. The test stimulus is comprehensive for the designer to reliably determine if connections among modules in the design are correct as intended or not.

In keeping up with test generation terminology, we use the word *fault* to refer to a possible alternative connection between two ports among interconnected modules in the given circuit. However, one or more connections of the given hierarchical circuit may be wrong, and one or more alternate connections, which are referred to as faults, might have been the design intention. The essential idea of fault detection in this work is to develop test stimuli that differentiates each single interconnection between two modules from its each alternative connection. Then the designer can determine whether the given circuit or an alternative is the right connection.

We identified five different fault types as the first phase. We also developed signal values needed to make the faults sensitive locally between module connections. In the second phase, software was written to create those signal values and propagate differential value pairs through the circuit to detect faults at the external boundary. The value pairs might be on a single signal line as in stuck at fault testing or on two different lines. We applied the software to benchmark circuits to obtain results on its effectiveness.

Section 2 of this paper describes the identification of alternate interconnections referred to as fault models in this work. It also explains signal value pairs required to sensitize alternatives (faults) with respect to the given gate level circuit. In Section 3 we briefly describe the implementation of the research work by software automation to compile, enumerate potential faults, create tests, and to simulate faults concurrently. Section 4 describes the nature of the circuits used for the experiment and includes summary of the data. Finally, the conclusion is given in Section 5.

2. INTERCONNECTION FAULT MODELS

As mentioned earlier, an alternative connection between two module instances may be the correct or the intended connection by the designer rather than the connection in the given circuit. We classify those alternate connections as faults. If an alternate connection between two ports is the intention, then their connection in the given circuit is in error. The most probable module interconnect errors can be classified into five different types, A through E. Together these classifications can account for nearly all module connection errors in hierarchical circuit designs. Additionally, having distinct classification allows them to be analyzed individually in order to determine the best way to distinguish the alternatives, i.e. detect the faults. Modules at a lower hierarchical level are typically created by design automation tools and higher-level modules are assembled by designers. Even the lower-level modules may include other modules in addition to logic elements as shown in Figure 1.

Modules in Figure 2 are hierarchical modules, themselves containing instances of modules shown in Figure 1. The figures depict module names and their instance names which appear after the colon (:) such as k0 appearing in HA:k0 in Figure 2(b) which is a half adder module in its functionality. The circuit model shown in Figure 3 illustrates the highest level (top level) of a circuit that will be used as an example to help depict the five different fault types described in this work. We used the modules shown in Figure 1 and Figure 2 with additional logic elements to build the larger circuit named MIR in Figure 3. The EHA module used the HA module once as shown in Figure 2(b), and the MIR circuit also used an instance of HA module which is named *insha* in Figure 3. MIR circuit has ports A0, B0, A1, and B1 acting as external inputs while Y0, Y1, Y2, and Y3 signals act as external output ports. Multiple signal sets are represented as a vector for clarity such as {A0,B0,A1,B1} to represent the A0, B0, A1, and B1 signal set.

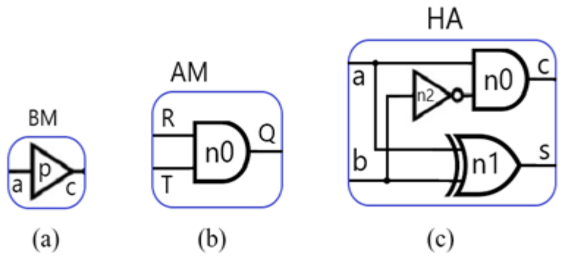

Figure 1. Example modules built with logic elements.

During the simulations a designer typically monitors the ports at higher levels of a circuit which may or may not be at the top level. In order to detect a potential error, the effect of that error, in the form of a pair of data values (desired value and actual value) must be made to appear at the erroneous locations. Here the actual value refers to the simulated value of the given circuit which may be incorrect if an alternative module connection is the correct one. Those value pairs then must be propagated via simulation to the circuit boundary being monitored to collect simulation data. In this work, we need to see values on alternate connections that are different from values on the connections of the given circuit. By originating that discrepancy on the possible misconnect site and propagating it further toward the circuit boundary, we can detect the fault.

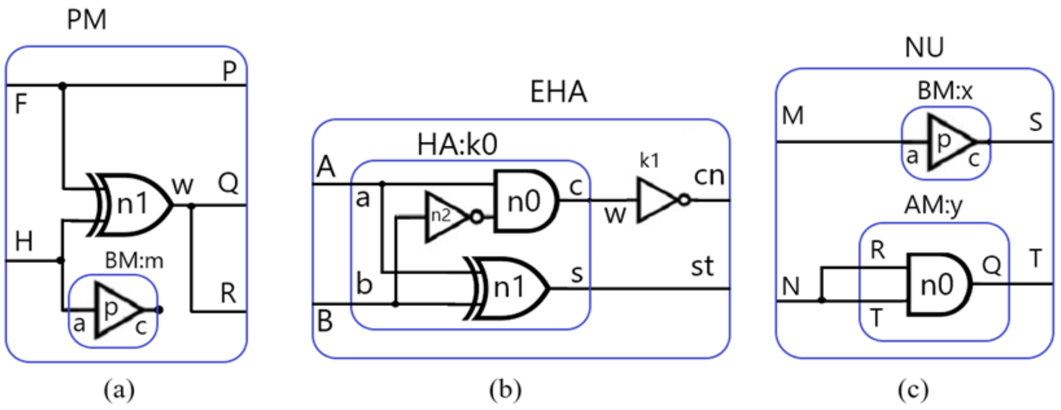

Figure 2. Example modules built with predesigned modules and logic elements.

2.1. Type-A Fault Model

A module integration fault of Type-A occurs when either an input port or output port of a module instance integrated as a component in building another module does not have a connection to a wire or a port that provides a driving signal. Figure 4 illustrates this as a separate instance of AM module used in Figure 3. If the circuit is described in a text-based hardware language (VHDL, Verilog, etc.) and processed using design automation tools, the initial compiling stage will detect any faults of this type and therefore no further exploration is needed for this type of potential errors.

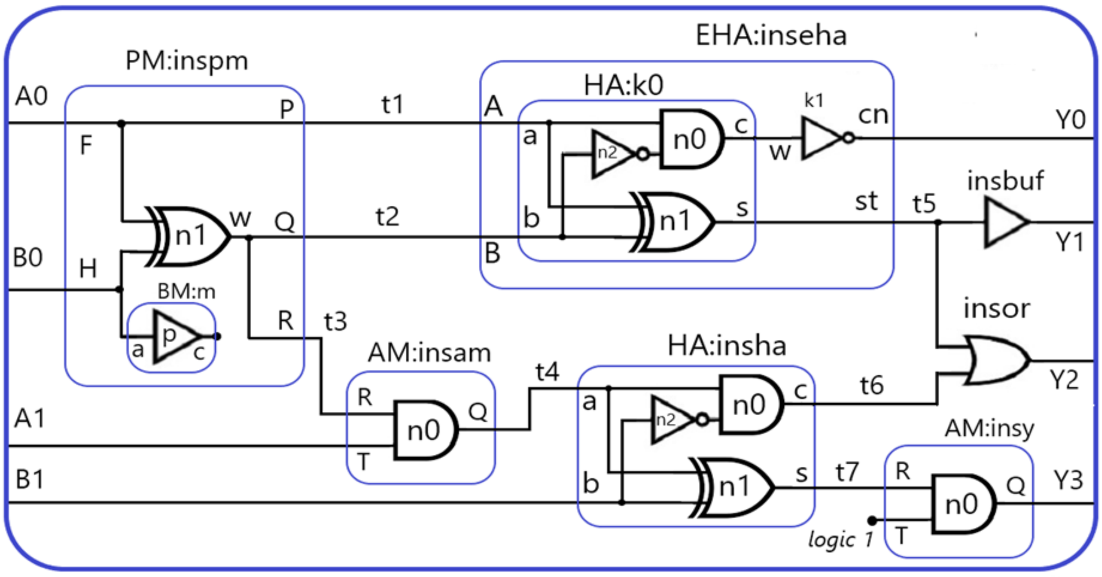

Figure 3. Top level circuit (MIR) built with other modules and logic elements.

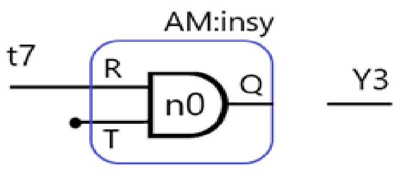

Figure 4. Type A fault – Unconnected ports. Port T and Q are not connected.

2.2. Type-B Fault Model

This error may occur when two or more output ports or internal signals are driven by a single port of a module instance. An example occurs in the output of PM module in Figure 2 at Q and R ports. It may occur in conjunction with Type-A faults depending on the module makeup of the circuit. Figure 5 shows the alternative connections where (b) and (c) yield two distinct circuits. In order to evaluate the potential error (fault) of port *c* of module *m* inside PM not driving t3 signal via the signal R. To detect that fault, a test stimulus producing opposing values, 01 (or 10) on the {t2,t3} pair is needed. This requires port *c* and signal *w* to receive 01 (or 10) respectively, to sensitize the potential faults. The differential effect of 01 vs 10 (or 10 vs 01) on {t2,t3} signal pair must be propagated through the circuit using external test vectors as typical in test patten generation. This validation may be extended by having alternative connections to any output port from the unconnected signal which produces three alternative (faulty) circuits for this example.

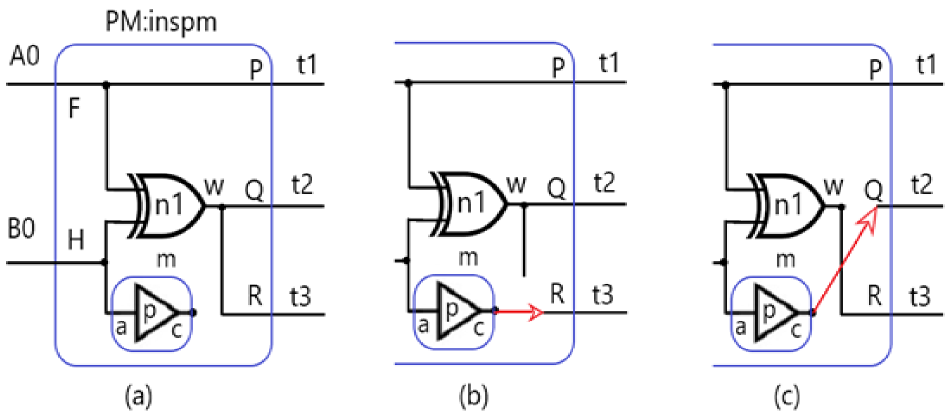

Figure 5. Type B fault – a single signal drives multiple ports with a signal left unconnected.

2.3. Type-C Fault Model

A fixed logic value, either 1 or 0, at an input port of a module instance is considered a potential integration error and categorized as a Type-C fault. This configuration is illustrated in Figure 6 with the AM module that is also in the MIR circuit shown in Figure 3. Detecting whether this is intent, or mistake requires originating the opposite value at the fault location and propagating it through. Similar to Type-A faults, Type-C faults will generally be flagged by software automation tool developed in the initial compilation stages.

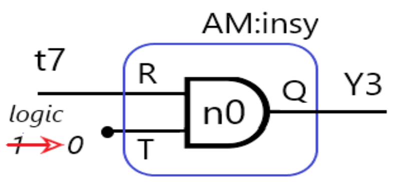

Figure 6. Type C fault – flipped logic constant.

2.4. Type-D Fault Model

A module integration fault of Type-D occurs when adjacent ports of a module are swapped with respect to the intended connections. Figure 7(a) depicts the interconnections of the *insha* instance of the HA module in the circuit of Figure 3. In the given circuit, the signal pair {t4,B1} is connected to {a,b} pair, and {c,s} pair is connected to {t6,t7} pair. Figure 7(b) contains two potential Type-D faults, one occurring at the input ports of swapped {B1,t4} to {a,b} pair, and the other at the output ports of swapped {c,s} to {t7, t6} signal pair.

Detecting a Type-D fault requires placing opposing pairs of values, 01 (or 10), at the corresponding swapped ports on {t6,t7} pair which is similar to Type-B fault detections. These ports may be input or output ports of a module or at the topmost boundary. The swapped pair of connections (the fault) is created virtually within the simulator to carry out the simulation of the given circuit and the alternative as two different circuits. This is a typical concurrent simulation of differential circuits without creating two completely alternative circuits and simulating the entire circuit for each enumerated fault.

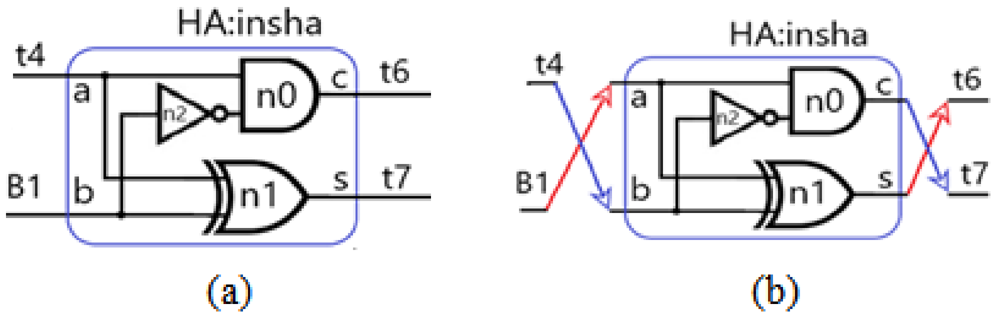

Figure 7. Type D fault – swapped port pairs on a module boundary.

2.5. Type-E Fault Model

When a set of signals outside of a module is being connected to the ports of that module, the designer may inadvertently create a wrong connection. While it is possible the erroneous connection may be random, we consider it to be a shifted connection for simplicity. This situation is classified as a Type-E fault. A Type-E fault occurring on only two ports may be reduced into a Type-D fault. Figure 8(a) shows three single line interconnections going from {P,Q,R} output ports of PM module to two other modules via the signals marked {t1,t2,t3}, respectively, which feed input ports of two other modules, EHA and AM. Figure 8(b) shows the interconnection between the three output ports of PM and the 3 {t1,t2,t3} signals in a different order which is shifted one position down relative to the connections in the given circuit, Figure 8(a). When the upward positional shift is considered, it will yield an additional alternative connection (fault). The goal of this test is to provide a stimulus which delivers 3 different results at the monitored boundary of the circuit, indicating that two alternative faults were detected and allowing the designer to determine which configuration was intended. Each test vector applied to local Type-E sites must consider the values on all three signals (or ports) involved in Type-E errors so that the fault effect is identified as a shifted value pair relative to the given circuit connections.

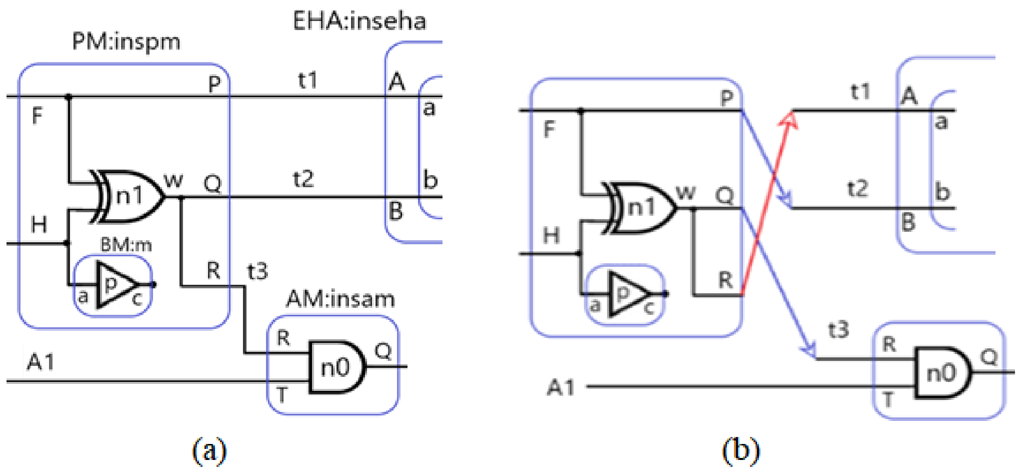

Figure 8. Type E fault – port connected with a positional shift.

In Figure 8(b), having 010 (or 101) appear on {P,Q,R} triplet results in a sensitized fault such that {t1,t2,t3} triplet would receive either 001 (or 110) due to the fault depicted in the figure. These particular shifted connections would generate a differential value pair of 010/001 (or 101/110) on the {t1,t2,t3} triplet. This type-E fault thus creates two signals with differential (fault effect) values. If the shifted connections of Figure 8(b) were in the other direction ({P,Q,R} going to {t3,t1,t2}, respectively), differential value pair on {t1,t2,t3} triplet would be 010/100 (or 101/011). Both of these will be considered since it is not apparent which connection – the given form in (a), shifted as shown in (b), or shifted opposite way (not shown)- is the right connection.

For a type-E fault to be detected both fault effects must be marked as detected at the top-level circuit boundary. If only one such fault effect is detected, then it may be only type-D in which just two signals got swapped. For type-E, two differential pairs tagged with the type-E fault location and the differential signal values must be detected at the top circuit or at the monitored boundary - not necessarily in the time cycle, if simulated sequentially.

3. Software Implementation

A software application named *Module Integration Error App* (*MIEApp)* was developed in C++ programming language to automate the module interconnect fault detection process for hierarchical single-bit circuits of any size. The step (i) shown in Figure 9 reads the circuit description in a simplified Verilog hardware description language format and compiles to build a computer in-memory database with the module hierarchy intact so that module boundaries are directly available for computations. Step (ii) inspects the module boundaries and their port interconnections to enumerate Type-B, Type-D and Type-E faults which are the most possible alternative connections. The tool directly identifies Type-A and Type-C faults during the circuit compilation step. It is noted that Type-E is created by potentially misplaced signal connections which are based on how the hierarchical database was created and therefore it may be slightly different from shifted signal sequences shown on the circuit portions in Figure 8. In step (iii), MIEAppselects a new undetected fault from the enumerated list. It then determines the test value pairs appropriate for the fault type at the local interconnect site to sensitize the fault as explained earlier. The value pairs must be such that the selected alternate connection and the connection in the given circuit will have opposite values. Step (v) is to create a test vector starting from the inputs to make the value pairs appear at the selected fault site.

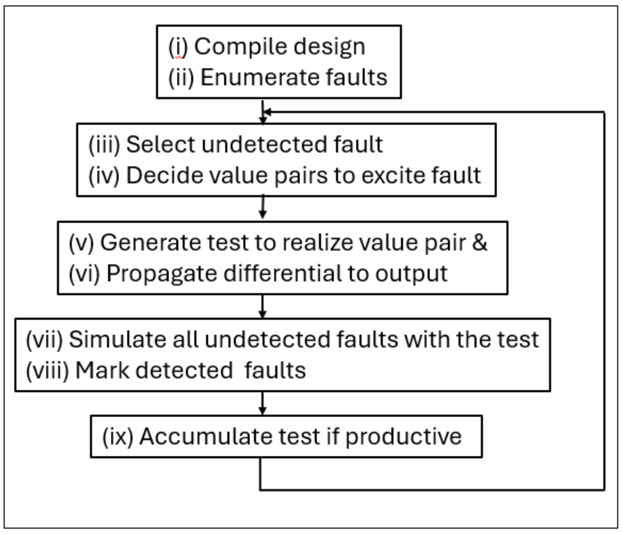
(i) Compile design
(ii) Enumerate faults
(iii) Select undetected fault
(iv) Decide value pairs to excite fault
(v) Generate test to realize value pair &
(vi) Propagate differential to output
(vii) Simulate all undetected faults with the test
(viii) Mark detected faults
(ix) Accumulate test if productive

Figure 9. The software process flowchart

In step (vi), the differential value pairs of the selected fault are propagated toward circuit outputs. Steps (v) and (vi) create a single unified test vector starting from the top-level input boundary by adopting the typical automatic test generation techniques [9]. Once a complete test vector has been developed it is simulated, without considering faults, on the compiled circuit model kept in the data memory of MIEApp. Additionally, in step (vii), each test vector generated by the tool is applied and simulated concurrently to all the undetected faults that were enumerated during the compilation stage. When a fault is detected, it is marked and removed from further consideration as is typically the case in test generation techniques. The fault simulation in MIEApp uses concurrent and differential faulty simulation algorithms [10] embedded into an event driven simulation engine. Time delays on the logic gate elements are taken as unit delays.

MIEapp tool is robust enough to apply on large circuits as evident from the results obtained. The logic simulation accuracy of the tool was validated on numerous circuits using the downloaded and locally built version of Icarus Verilog software [11]. The MIEApp tool does not analyze the data flow paths in a circuit. Therefore, it enumerates many Type-D and Type-E faulty alternatives for circuits using signals and ports used for interconnections among module instances. While the accuracy of the logic simulation of MIEApp was validated by manually comparing signal values against simulated values in Icarus simulator [11], no other software was available publicly to compare the fault simulation accuracy or performance against other similar software tools. Therefore, the accuracy of fault simulations in MIEApp was verified manually on several small circuits for Type-C, D and E faults both through software debugging traces and on paper.

4. TEST CIRCUITS AND RESULTS

The software tool, MIEApp was developed under the Microsoft Visual Studio C++ 2019 version 16.11. MIEApp was applied on several circuits which are briefly described below. Many small circuits were used to verify the logic simulation accuracy followed by the fault detection accuracy for B, D and E fault types. Some of the ISCAS 85 benchmark circuit descriptions (publicly available) accompanied a vector set [7] developed for stuck at fault testing. These circuits were simulated with the test vectors on both MIEApp and iVerilog [11]. The logic values appearing at the outer boundary of the circuits matched between the two simulators for every ISCAS circuit reassuring the accuracy of MIEApp.

MIEApp uses any external stimuli vector provided and retained only those that were productive in detecting any faults. If faults are still left undetected, the tool generates test stimuli vectors and screens each by simulating against the remining faults before accepting the vector as useful. The tool ran on a Windows 10 desktop with 8 GB RAM and 2.2 GHz AMD A6 processor supported by 512 GB SSD drive. Only the ISACS 6288 circuit was evaluated under a different Windows 10 desktop with 3.6GHz i5 processor combined with 8 GB RAM and an optical drive due to its larger size.The Verilog grammar-based text descriptions for each of the ISCAS-85 circuits used are relatively long and not included in this paper. Further information for each of the circuits can be found in public domain [7, 8]. Again, it is important to note that each of the hierarchical Verilog descriptions were converted into a single bit-oriented representation for MIEApp.

4.1. Circuit ME

This circuit shown in Figure 10 was primarily used as a test case to verify the accuracy of MIEappin every aspect of this project due to its small size. After the unit testing of the software, this circuit validated the MIEApp steps in compiling and building the hierarchical data base, accuracy of the event driven unit delay logic simulation, correctness of the differential and concurrent fault simulation with tracking of differential data pairs for Type-D and Type-E faults, and the vector generation step followed with the internal simulation.

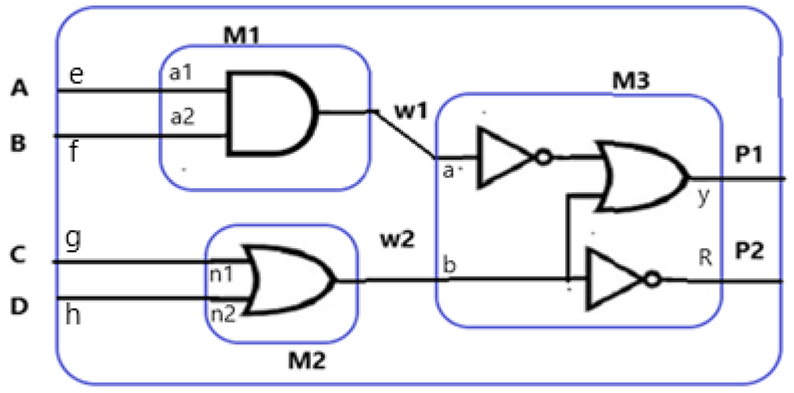

Figure 10. The verification circuit, ME

MIEApp detected 6 out of 8 Type-D faults it enumerated in the circuit shown in Figure 10 generating 3 test vectors. It was able to detect all the 6 Type-E faults it enumerated using the same 3 vectors. Due to the symmetry of input signals to modules M1 and M2, swapped alternatives of {A,B} and {C,D} are not distinguishable. The 3 test vectors were 1100, 1010 and 0101 for {A,B,C,D} inputs. Each fault is tagged with a numerical identification number for internal reference by the software. Type-E faults require a triplet of ports on the module boundary, and only {A,B,C,D} in ME of Figure 10 can result in alternate Type -E faults.

Table 1 provides the alternate connections enumerated and maintained by MIEApp as differential circuit segment for each of those Type-E faults. In the given circuit, {A,B,C,D} external signals feed {e,f,g,h} ports of ME in Figure 10. The second row of Table 1 shows the Type-E faults which are the possible alternate connections of external {A,B,C,D} inputs going into ME circuit.

The last row shows which test vector on {A,B,C,D} distinguishes the alternatives (faults) from the connections given in the circuit. Table 2 provides a summary of the output results showing the enumerated list of the faults and their detection status per effective test vector.

Table 1. Enumerated Type-E faults of ME circuit.

Fault ID	9	10	11	12	13	14
Alternative connections	{A,B,C,D} = f,g,e,h}	{A,B,C,D} ={e,g,h,f}	{A,B,C,D} ={g,f,h,e}	{A,B,C,D} ={e,h,f,g}	{A,B,C,D} ={g,e,f,h}	{A,B,C,D} ={h,e,g,f}
Vector	1010	1100	0101	1100	1100	0101

Table 2. Summary of detection status of Type D and Type E faults of ME circuit.

Vector on {A,B,C,D}	Fault description from MIEApp on its output report	Detected at port
1100	ID= 6: Type D: /.B drives M2.n2 instead of /.D AND /.D drives M1.a2 instead of /.B	P2
	ID= 4: Type D: /.A drives M2.n2 instead of /.D AND /.D drives M1.a1 instead of /.A	
	ID= 5: Type D: /.B drives M2.n1 instead of /.C AND /.C drives M1.a2 instead of /.B	
	ID= 3: Type D: /.A drives M2.n1 instead of /.C AND /.C drives M1.a1 instead of /.A	
	ID= 8: Type D: /.w1 drives M3.b instead of /.w2 AND /.w2 drives M3.a instead of /.w1	
	ID= 12: Type E: /.D drives fanouts of /.C, which drives /.B fanouts	
	ID= 10: Type E: /.B drives fanouts of /.C, which drives /.D fanouts	
	ID= 13: Type E: /.C drives fanouts of /.B, which drives /.A fanouts	P1
	ID= 1: Type D: M3.y drives /.P2 instead of M3.R AND M3.R drives /.P1 instead of M3.y	
1010	ID=9: Type E: /.A drives fanouts of /.B, which drives /.C fanouts	P2
0101	ID= 11: Type E: /.C drives fanouts of /.D, which drives /.A fanouts	P2
	ID= 14: Type E: /.B drives fanouts of /.A, which drives /.D fanouts	
Not detectable	ID= 7: Type D: /.C drives M2.n2 instead of /.D AND /.D drives M2.n1 instead of /.C	
	ID= 2: Type D: /.A drives M1.a2 instead of /.B AND /.B drives M1.a1 instead of /.A	

4.2. Circuit MIR

This circuit in Figure 3 contains basic logic modules that were used to help visualize and develop the five different module interconnection fault types seen in figures 4-8. This MIR circuit was used extensively for the development of methods to categorize various interconnection errors to facilitate their detection by software automation.

MIEApp created 3 Type-B faults considering that the unconnected signal in PM module may drive one of the three output ports yielding 3 alternate connections. They were all detected by the software tool. The single undetected Type-D fault was due to the swapping of {Q,R} signals which cannot be detected since the same signal feeds both of them. Out of the 28 Type-E faults, only 18 were detected which implies that MIEApp created test vectors to distinguish 18 different connections between modules alternative (faults) to the connections in the given circuit. Ten of the Type-E alternate connections cannot be detected or distinguished due to certain fan out configurations such as t2 and t3 signals having the same value, Y1 and Y2 have shared signals, and the *insy* instance of AM has a fixed input value.

4.3. ISCAS 74283 Circuit

This combinational adder circuit presented with ISCAS benchmarks [7] consists of 3 different modules as shown in Figure 11. The GP module generates parity signals of A and B inputs allowing CLA module to generate carry out signals from the addition operation. The final SUM module provides the results of the addition of A and B combined with the single carry in, C_0.

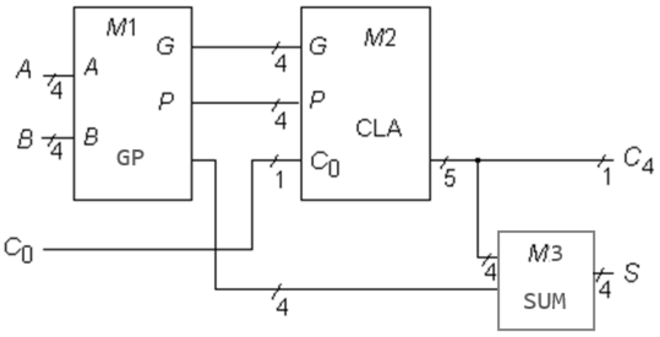

Figure 11. ISCAS 74283 Four-Bit Adder circuit

4.4. ISCAS 74L85 Circuit

The 4-bit magnitude comparator [7] had 3 control inputs and 4-bit data inputs for comparison. It produces 3 output signals based on the control input values. It is functionally modelled as shown in Figure 12 with 4 separate modules for this research work. The CLA module is a carry look ahead adder and this circuit contains two of its instances. The GP module generates signals for CLA modules based on the two control signals checking for inequality. The EQ module is the simplest with a single gate. The circuit yields a total of 467 Type-D and Type-E alternative interconnections (faults) among modules, inputs, and output signals.

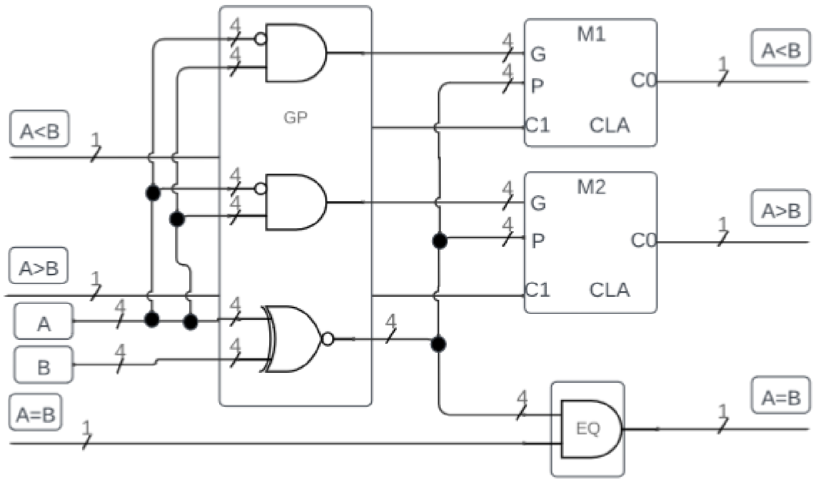

Figure 12. ISCAS 74L85 - Four-Bit Magnitude Comparator Circuit

4.5. ISCAS C432 Circuit

The C432 circuit shown in block diagram from in Figure 13 is an interrupt controller for 3 multichannel bus groups. Each interrupt request bus (labeled A, B, and C) contains 9 individual channels, where the bit position within each bus determines the interrupt request priority. The input bus of 9 bits labeled E enables and disables interrupt requests within the respective bit positions. The request in A has higher priority over B, which has priority over any request from the B group. Within each bus, the position of each channel relative to the others gives higher or lower priority to the request. The circuit in Figure 13 consists of unique individual modules labeled M1, M2, M3, M4, and M5, which contain the underlying logic gates. The seven outputs PA, PB, PC and the four bits in Chan output signal specify, as a binary coded decimal value, which channels have acknowledged the interrupt requests from A, B, or C. The logic in module M5 also provides a 9 to 5 priority encoder. All the outputs, except Chan, are single bit signals while all the inputs and all the other internal signals are 9 bits wide. This circuit yielded over 5000 possible alternate interconnections among the five modules, inputs, and output signals.

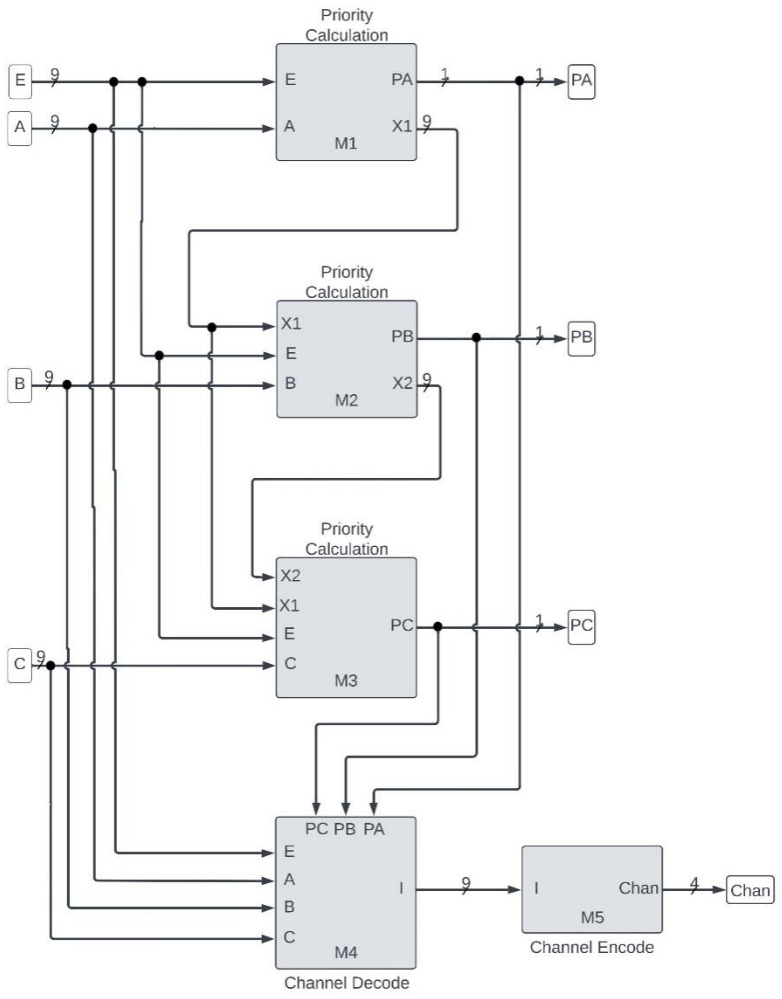

Figure 13. ISCAS C432 - 27- Channel Interrupt Control Circuit

4.6. ISCAS C6288 Circuit

This circuit is a 16 x16 bit multiplier which is consisting of 225 full adders and 15 half adder modules arranged in a matrix. The C6288 circuit has two 16-bit signals as inputs and a 16-bit output bus. To obtain module interconnections, the circuit modules were arranged as a 15x16 matrix providing a two-level hierarchy. The circuit is Figure 14 is a reduced representation of the large original circuit showing only a 4x4 multiplier. In that, modules labelled F are full adders and those with H are half adder modules. With a shallow hierarchy and many module instances, MIEApp enumerated over 250,000 potential alternative interconnections of Type-D and Type-E.

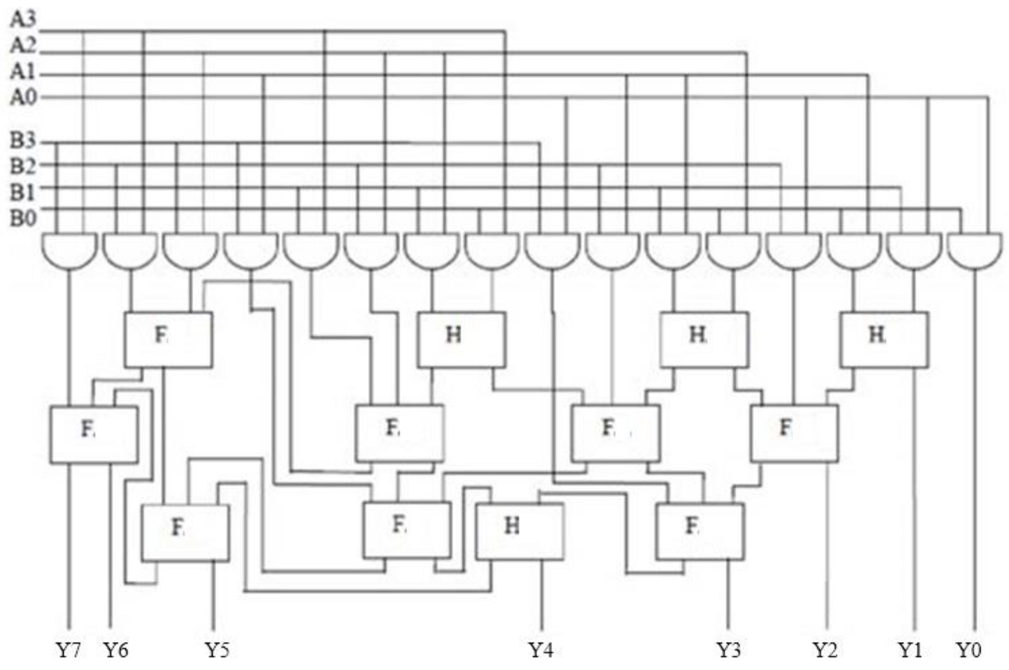

Figure 14. Reduced model of ISCAS C6288 circuit

Table 3. Summary of results from MIEApp on the benchmark and other circuits.

Circuit	**Runtime (minutes)**	**MI count detected/total (%)**	**Vectors created**	**Type-B count detected/ total**	**Type-D count detected/ total**	**Type-E count detected/ total**
ME	0.0437	12 / 14 (86%)	3	0/0	6/8	6/6
MIR	0.0290	37/ 48 (77%)	8	3 / 3	16 / 17	18/28
74283	0.1265	406 / 406 (100%)	20	0/0	238 / 238	168 / 168
74L85	0.1247	467 / 467 (100%)	35	0/0	299 / 299	168 / 168
C432	312	5220/5334 (98%)	56	0/0	4680 / 4680	540 / 654
C6288	3582	250755/251146 (99%)	334	0/0	247696/247696	3059/3450

5. CONCLUSION

To reduce the effort needed to verify the accuracy of the module integration step of digital circuit design, we developed five fault models representing the common module interconnection errors in the initial stage of this work. We formulated logic value pairs needed to make alternate connections or faults to show signal value differentials at the locality where faults were enumerated. With typical test generation techniques enhanced to target differential value pairs, we were able to generate tests and simulate them in a concurrent event driven simulation that we implemented in C++ as an integrated software tool, MIEApp. After a reasonable level of debug and error checking we applied the software to several circuits including ISCAS benchmark circuits to obtain results that were presented here. The main limitation was that the circuits were constrained only to combinational circuits. The generated test vectors distinguished (detected) in a range of 77% to 100% of all potential alternative module interconnections in the tested circuits listed in Table 3. We were able to apply the software onto a large circuit like C6288 to show the robustness of the algorithms and the implemented MIEApp software tool.

REFERENCES

[1] S. Kang and S. Szygenda, (1992) “Automatic Error Pattern Generation for Design Error Detection in a Design Validation Simulation System”, Proceedings of 5th Annual IEEE International ASIC Conference, pp. 533-536.

[2] S. Kang and S. Szygenda, (1992) “New Design Error Modelling and Metrics for design Validation”, proceedings of Euro-DAC ‘ 92, pp. 472-473.

[3] Shi-Yu Huang, et. al, (1998) “Fault-simulation based design error diagnosis for sequential circuits”, Proceedings of 35th Design Automation Conference, pp. 632-637.

[4] Kai-hui Chang, (2007) “Functional Design Error Diagnosis, Correction and Layout Repair of Digital Circuits”, University of Michigan, PhD Dissertation.

[5] S. Almukhaizim, et. al, (2003) "On compaction-based concurrent error detection," *9th IEEE On-Line Testing Symposium, IOLTS 2003.*, Kos, Greece, pp. 157-162.

[6] S. Kavitha, et. al, (2019) “A New Approach of an Error Detecting and Correcting Circuit by Arithmetic Logic Blocks”, International Journal of Electronics and Telecommunications, VOL. 65, pg. 313-318.

[7] https://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html

[8] https://pld.ttu.ee/~maksim/benchmarks/

[9] P. Goel and B. C. Rosales, (1981) "PODEM-X an Automatic Test Generation System for VLSI Logic Structures", Proceedings of 18th Design Automation Conf., pp. 260-268.

[10] S. Gai and P. L. Montessoro, (1991) "Creator: General and Efficient Multilevel Concurrent Fault Simulation", Proceedings of 28th Design Automation Conference, pp. 160-163.

[11] https://sourceforge.net/projects/iverilog/

AUTHORS

Nicholas Dematteis, **Jesus Godinez, and Gina Rhoads** are senior level undergraduate students pursuing bachelor’s degree in computer engineering at the engineering and computer science division of Pitt-Johnstown campus of the University of Pittsburgh, Pennsylvania, USA. All three graduated in April 2024.

MADDU KARUNARATNE earned his Ph.D. degree in electrical engineering from the University of Arizona, Tucson, AZ. Before joining Pitt-Johnstown in 2004, he gained fourteen years of industry experience working in the semiconductor industry performing software development, application engineering, design, testing and verification of digital integrated circuits. His research interests are in design and test automation, power analysis, and software development. He has authored numerous publications and holds several US patents.