uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Mohammed Elbadri
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Reconfigurable Processing Unit for Digital Circuit Testing Using Built-In Self-Test Techniques

TITRE DE LA THÈSE / TITLE OF THESIS

V. Groza
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

M. Bolic

R. Goubran

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A RECONFIGURABLE PROCESSING UNIT FOR DIGITAL CIRCUIT TESTING USING BUILT-IN SELF-TEST TECHNIQUES

By

Mohammed Elbadri

B.A.Sc., University of Ottawa

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies of the

University of Ottawa in partial fulfillment

of the requirements for the degree of

Master's of Applied Science

Ottawa-Carleton Institute of Electrical and Computer Engineering

School of Information Technology and Engineering

Your file Votre référence
ISBN: 978-0-494-34067-7
Our file Notre référence
ISBN: 978-0-494-34067-7

ABSTRACT

Digital circuit testing is presented in this thesis. This thesis introduces an architecture that accelerates benchmarked circuit testing. Traditionally, benchmark circuits are tested on software, because of the complexity in developing a generic hardware architecture capable of testing sequentially or concurrently. The testing is based on Built-in Self-Test (BIST) techniques. The circuit testing is accomplished by two hardware implementations, aimed at increasing execution time with respect to its counterpart, software. The implementation realized and executed in hardware, illustrates the advantages of utilizing hardware platforms for digital circuit testing and it gives a path to developing more complex benchmarked circuits; which would have higher fault coverage.

The novel architecture is targeted for digital circuit testing and adaptive embedded system applications. These applications vary in their constraints (i.e. hard and soft real-time constraints) and environmental conditions (i.e. unknown and unpredictable).

The novel architecture consists of a fixed hardware unit and a Reconfigurable Processor Unit (RPU). The RPU employs hardware functional blocks. These Hardware Blocks (HB) encompass logic that targets their respective applications. We realize and implement HBs that target digital circuit testing applications, by means of BIST techniques.

Experimental results are presented in this thesis. In simple terms, the speedup factors are as high as 1×10^4 for sequential testing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my thesis supervisor Dr. Voicu Groza for his help, constant support, and patience throughout my Masters.

I would like to extend a special thanks to Rami Abielmona for his ever-lasting push to the finish line, and thanks for his help with everything that everyone had to know and had to do when they first started graduate studies, he was always there for help, never let me down. I owe him a lot for everything I struggled through, especially for helping me see the light at the end of the tunnel. Thank you for everything, especially the FPSoc idea that led me to pursue a Master's degree. As well, a thank you is owed to my team and guest members, Mohammad El-Kadri, Arkan Khalaf, and Dr. Mansour Assaf.

Exceptional and particular thanks go to my parents, brothers (Mahmoud and Ahmed), and sisters (Ala'a and Riham) for their words of encouragements and their perseverance through rough times. As well, thanks to Özgür Ekici and Waleed Altoukhi for being like brothers to me and always reminding me of how to enjoy life. Last but not least a warm thanks to my very close friends as they have helped a great deal with my Masters, either it be technically or not.

LIST OF ACRONYMS

A/D	Analog-to-Digital
AE	Application Element
AF	Application Flow
AML	Application Modelling Language
ASIC	Application Specific Integrated Circuit
ATE	Automated Test Equipment
BIST	Built-In Self-Test
CA	Cellular Automation
CF	Compact Flash
CMC	Canadian Microelectronics Corporation
CMOS	Complementary Metal-Oxide Semiconductor
CODEC	COder-DECoder
CPLD	Complex Programmable Logic Devices
CTR	Compile-Time Reconfiguration
CUT	Circuit Under Test
D/A	Digital-to-Analog
DAC	Digital-to-Analog Converter
DFT	Design For Test
DIP	Digital Image Processing
DIP switch	Dual In-line Package switch
DSP	Digital Signal Processing
DTPG	Deterministic Test Pattern Generator
EDK	Embedded Design Kit
EMI	Electro-Magnetic Interference

ERACE	Embedded Research Architecture for Co-design Environment
ETPG	Exhaustive Test Pattern Generator
FIM	Fault Injection Multiplexer
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GEMS	Group for Embedded MicroSystems
HB	Hardware Block
HDL	Hardware Description Language
IC	Integrated Chip
ICAP	Internal Configuration Access Port
IMTC	Instrumentation and Measurement Technology Conference
ISCAS	International Symposium on Circuits And Systems
JIT	Just-In-Time
JTAG	Joint Test Action Group
LM	Local Memory
LUT	Look-Up Table
MAB	Memory Access Bus
MAC	Media Access Control
NTSC	National Television System Committee
OM	On-chip Memory
OPB	On-board Peripheral Bus
PAL	Phase Alternation Line
PC	Personal Computer
PLD	Programmable Logic Devices
RAM	Random Access Memory
RcP	Reconfigurable co-Processor
RE	Reconfigurable Element
RF	Reconfigurable Flow

RGB	Red Green Blue
RNG	Random Number Generator
RPU	Reconfigurable Processing Unit
RS-232	Recommended Standard-232
RTPG	Random Test Pattern Generator
RTR	Run-Time Reconfiguration
SVGA	Super Video Graphics Adapter
TPG	Test Pattern Generator
VHDL	VHSIC-HDL
VHSIC	Very-High-Speed-Integrated-Chip
WISES	Workshop on Intelligent Solutions in Embedded Systems
ZBT	Zero Bus Turnaround
μB	MicroBlaze

Chapter 1

1. Introduction

This thesis introduces an architecture that targets circuit testing on a reconfigurable architecture. The proposed architecture is built on a soft-core fixed microprocessor and a Reconfigurable Processor Unit (RPU). The former guarantees meeting soft and hard deadlines in order to achieve the intended system functionality, while the latter holds different functional Hardware Blocks (HB). The HBs can target different applications, such as digital circuit testing, Digital Signal Processing (DSP), and Digital Image Processing (DIP). This thesis targets digital circuit testing applications. The reconfigurable digital tester that has been conceived and developed in this project is capable to running various algorithms for Built-In Self-Testing (BIST) of digital cores at speeds which are higher with several orders of magnitude than the current software approaches.

1.1. Failures and Faults

As stated in [1], a failure occurs in a circuit if its operation differs from its specified behaviour. However, a fault is a physical defect that may or may not cause a failure. A fault can either be a logical or a non-logical fault. The former causes the logic value at any point in the circuit to become opposite to the specified value. The latter occurs by other events such as power failure, malfunction of the clock signal, etc.... Logical faults can be either localized or distributed. A local fault only affects a single variable (for example the logical output), while distributed fault affects more than one variable (for example a

clock malfunction is a distributed fault). Another measure of faults is its duration, weather the fault is a permanent or temporary fault. A temporary fault could be caused by Electro-Magnetic Interference (EMI), as one example, or any other transient faults or nonrecurring temporary faults. Meanwhile, permanent faults are recurring faults that are not a result of loose connections or other intermittent faults.

Roughly 44% [1] of total faults are caused by specification faults and design rule violations, with exception to intermittent faults. Therefore, we must model for such faults and detect these faults. By representing the effect of a fault through a fault model, we observe the change the fault generates in the circuit signals. There are several models that target different faults. The following fault models are most commonly used:

1. Stuck-at fault
2. Bridging fault
3. Stuck-open fault

1.1.1. Stuck-at fault

The stuck-at model is often referred to as the classical model, because it proffers a depiction of the most common types of failures. This model assumes that a fault in a logic gate results in one of the gate's inputs, outputs, or inter-wires being fixed to either a logic 0 (stuck-at-0) (Ground) or a logic 1 (stuck-at-1) (V_{source}). Stuck-at-0 and stuck-at-1 are usually referred to as s-a-0 and s-a-1, respectively.

To clarify the stuck-at fault model used, lets consider an example of a two-input, one-output digital circuit defined by the Boolean function: C = A + B (Figure 1). Assume that the second input (B) of the OR gate is unintentionally, by nature of the faulty gate, is connected to logic 1 instead of the input terminal B. Input B with a stuck-at 1 (s-a-1) is causing the fault in the circuit, i.e., regardless if the value of A and B are zero, output C will always be stuck to logic 1.

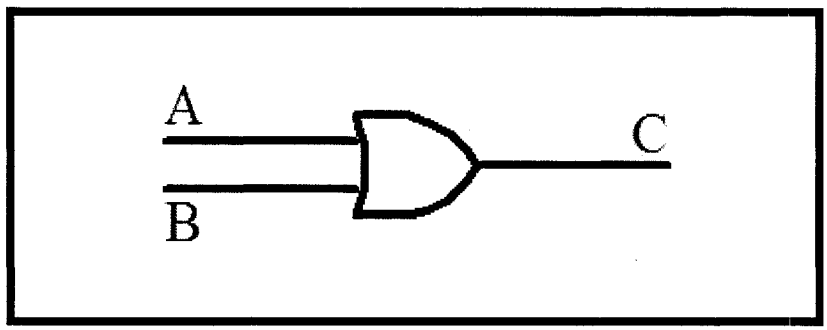


Figure 1: OR Gate

Representation of a s-a-1 on terminal B of the OR gate is a very helpful and useful fault model because it makes testing the circuit (OR gate) easier. However, stuck-at faults cannot model all kinds of faults especially physical defects of a Complementary Metal Oxide Semiconductor (CMOS).

1.1.2. Bridging fault

Bridging faults cannot be modeled as stuck-at faults because these faults occur on lower levels than gate-logic. Such as transistor level faults that have to be studied through CMOS realization of the gate/function. Bridging faults are related to physical faults/defects in semiconductor or layout. There are three categories of bridging faults:

- Input bridging
- Feedback bridging
- Nonfeedback bridging

Input bridging conforms to the short-ing of a certain number of inputs into a function or gate. Meanwhile, feedback bridging implies a short between one of the output with the input of a function or gate. Lastly, a nonfeedback bridging fault is a bridging fault that does not fall in the scope of the two previous bridging faults.

1.1.3. Stuck-open fault

Stuck-open fault models on the transistor level; similar to the bridging fault model, this model encompasses models that cannot be modelled as stuck-at fault models. To test circuits for shorts and open at the transistor level we utilize this model where a short is represented by a stuck-on transistor and an open is represented by a stuck-open transistor. A stuck-on transistor means the short-ing (short) of the source and the drain of the transistor, where as a stuck-open transistor means an open-ing (open) of the source and the drain.

The extent of this thesis is to find logical-local-permanent faults. To represent these kinds of faults and the effects of these faults, we will utilize the "stuck-at fault" model, more specifically single stuck-at fault model.

It is worth noting that the faults are not always detectable for two reasons. Firstly, if the fault in a larger than 1 level circuit cancels with another fault, hence

not changing the value of the output. Secondly, faults are not always detected by observing the output of the circuit since the output will seem correct for all combination of the inputs other than the one mentioned above in the example. These types of undetectable faults are termed hidden faults.

1.2. Testing

The ever-increasing density of Integrated Circuits (IC) has increased the probability of fault occurrences in digital systems. These systems, which are designed to operate in the field for an extended period of time, require more efficient methods of testing to ensure their reliability. Due to the increasing complexity in the integration of core-based designs, it is essential to find highly efficient testing techniques that ensure correct system performance [2][3][4][5][6][7].

Verification is essential to the design flow. However, even after verification, simulation, testing, synthesis, and place & route, we would like to know how well our design's functionality is, and more importantly for us, if our design is fault-free. By introducing s-a-0 and s-a-1 faults we can determine if we have detectable and undetectable or hidden faults. The former is a measure of how well the circuit responds to fault injections, and the latter is a measure of how reliable the circuit is, since a hidden fault can give the impression that the circuit is correctly functioning, while in reality, it is not. These fault injection techniques for testing digital system have shown to be very efficient in studying circuit behaviour when faults occur. This is done by applying test patterns generated by a Test Pattern Generator (TPG) to the CUT and comparing the outputs to known correct outputs. The outputs that are not affected by a stuck-at fault, are the hidden

faults. The hidden faults need to be identified, and can be identified with the appropriate test pattern, hence it is imperative to select the appropriate test pattern. Nevertheless, pseudorandom test pattern generation is the most ideal TPG as it will be discussed in more details in subsequent chapters.

Now, this type of fault detection system needs to be implemented on a flexible, module, and scalable system, hence an RPU is essential and will be introduced in the following sections. First, let understands the evolution to the RPU.

1.3. Configurable Computing

Configurable computing refers to the usage of configurable integrated circuits such as Field Programmable Gate Arrays (FPGAs) or Complex Programmable Logic Devices (CPLDs) in order to execute tasks. The tasks are modeled as combination of logic elements and are programmed onto the configurable integrated circuit. For FPGAs the integrated circuit is represented by the FPGA's Look-Up Tables (LUTs) (i.e. a circuit's truth table is represented in a LUT). With configurable computing, the integrated circuits are configured once on an FPGA and they are executed in parallel, unlike software which executes sequentially.

The main advantage of such architectures is the ability to be configured more than once without going through re-manufacturing the integrated circuits. This greatly reduces the cost endured in the re-manufacturing of fixed circuits, namely, Application Specific Integrated Circuits (ASICs). However, ASICs remain faster in terms of execution time and are not constrained to be of a particular size as they are specifically manufactured to virtually any required size. Problematically, configurable architectures are inflexible because they are configured once and

then the architecture is fixed. However, reconfigurable architectures can be reconfigured after realization to modify their functions.

1.4. Reconfigurable Computing

The field of reconfigurable computing arose to overcome the obstacles posed by fixed architecture computing and configurable computing. Namely, these are the inflexibility of fixed architectures and the poor performance of configurable architectures.

Reconfigurable architectures utilize Programmable Logic Devices (PLDs), such as FPGAs or CPLDs. A reconfigurable architecture can be loaded with a new configuration of logic; however, when reconfiguring, FPGA halts its execution and waits for the new configuration for it to start execution once again.

With this method, the space limitation of configurable architectures is virtually eliminated because the architecture is reconfigured accordingly with the proper logic at any given time slice. As well, reconfigurable computing greatly increases the flexibility and modularity of the device in use. However, the price to pay is the overhead in reconfiguring the whole device.

Nonetheless, partial reconfiguration (i.e. the ability to configure part of the FPGA separately and independently of the rest of the FPGA) is achievable and beneficial to cut cost of overhead in reconfiguring the whole FPGA. However, partial reconfiguration is not useful if the rest of the FPGA is temporarily halted. Therefore, the most favourable architecture is to be able to partially reconfigure the FPGA while the rest of the FPGA is active, termed Run-Time Reconfiguration (RTR).

1.5. RunTime Reconfigurable Computing

Runtime reconfigurable computing provides the capability of reconfiguring part of the FPGA while the rest of the FPGA is active (i.e. one part of the FPGA may be reconfigured with different DSP blocks while the other part is configured with a processor that utilizes the different DSP blocks as they are reconfigured). This new paradigm is promising as our digital systems are ever-increasing in complexity. Runtime reconfiguration can deliver software-like operating system characteristics, such as task swapping, task scheduling, etc..., with the help of a processor. The aim is to minimize the overhead acquired from reconfigurable computing, and to harness the advantage of reconfigurable computing, which is eliminating the space limitation presented by configurable devices.

To take full advantage of RTR, a second processor was introduced to undertake the task of managing the runtime reconfiguration. Since the main processor was already part of our system (which will be explained in the subsequent chapters), the RTR-processor is referred to as the co-processor. However RTR is outside the scope of this thesis.

Nevertheless, to pave the way for RTR we propose an RPU that contains multiple hardware functional units that can be partially reconfigured by the co-processor once RTR is "up and running".

1.6. Reconfigurable Processing Unit

The RPU is a reconfigurable unit which encompasses static hardware functional units. Hardware functional units are referred to as HBs. Static from the sense

that once the RPU is executing its HBs, the RPU will not be reconfigured to add another HB, we just load all the necessary HBs at start-up, hence static HBs.

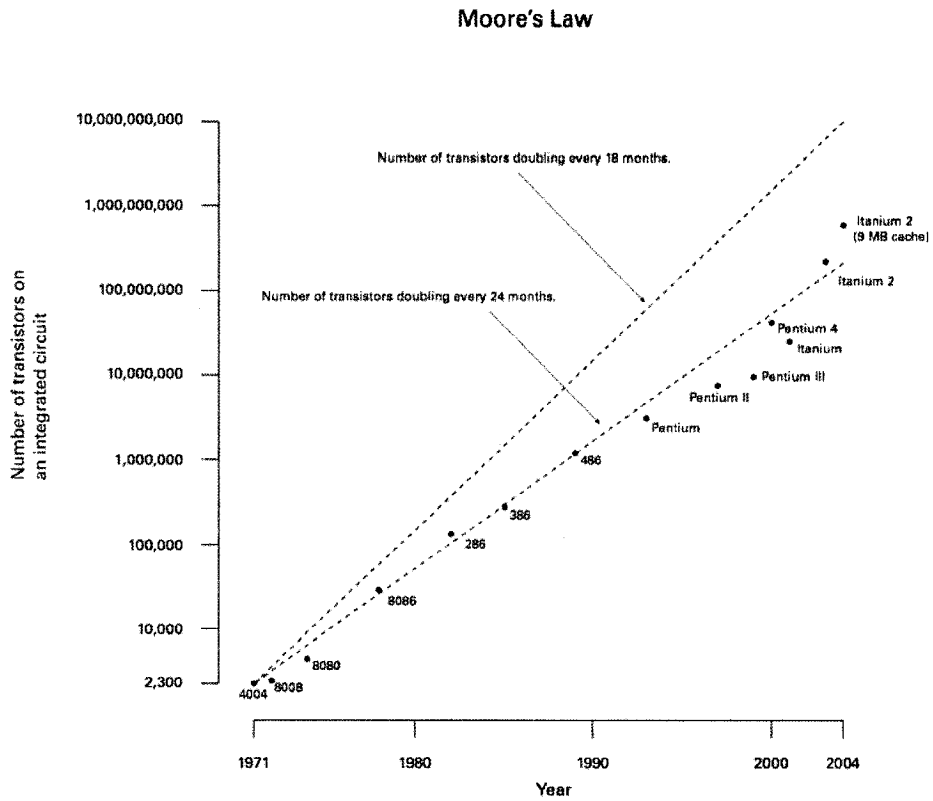
This thesis spawned from the Embedded Research Architecture for Co-design Environment (ERACE) project by the Group for Embedded MicroSystems (GEMS) Research Lab. The project introduces four competing realization of an algorithm used to catch both detectable and hidden faults with a CUT. The hardware implementations of the design test architecture and verification environment are suitable for Built-In Self-Testing (BIST) of digital cores, and build on the work presented in [2] and [8]. This thesis encompasses two of the four realization used to catch faults, namely Sequential and Parallel hardware testing.

1.7. Built-In Self-Test

With ever-lasting advancements in Very Large-Scale Integration (VLSI) technologies and the desire to satisfy Moore's law [9] of doubling complexity of Integrated Circuits (ICs) every 24 months, we observe that today's chips contain a very large number of transistors (Figure 2). To verify correct functionality of these complex chips, we must test these chips as thoroughly as we can. However, the cost of testing chips has grown dramatically for several reasons: (1) Increased complexity of ICs; (2) Design For Test (DFT) has become a distinct part of the chip development cycle, and; (3) Hourly charges for Automated Test Equipment (ATE) services has grown because of the time required to test the new ICs equipped with more transistors and more design complexity.

Therefore a testing technique was introduced to reduce design complexity, cost, and time required for testing. This is accomplished by reducing reliance upon

external (pattern-programmed) test equipment (ATE). Seeing as the test patterns and the expected responses of the Circuit Under Test (CUT) to these test patterns are utilized by an ATE to verify if the expected responses match the actual responses, we utilize a BIST for test pattern generation and evaluation of the output responses on the chip (embedded) to eliminate and/or minimize the use of the expensive ATE machines.

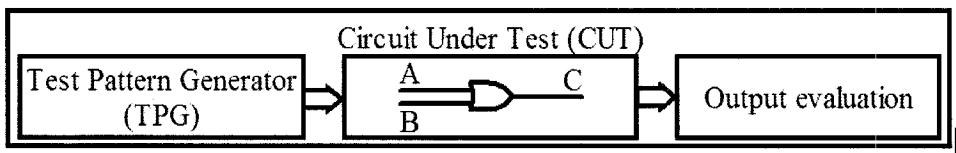


[9]

Figure 2: Growth of transistor counts for Intel processors (dots) and Moore's Law (upper line=18 months; lower line=24 months) [9]

Figure 3 presents a basic BIST structure[1]. BIST encompasses three main functions. The Test Pattern Generator (TPG) is responsible for generating and

applying test pattern to the CUT. Secondly, the CUT is what is the circuit being tested. Lastly, the output evaluation function is responsible for comparing the expected responses with the actual response of the circuit. Ideally, a BIST system is easy to implement and offers high fault coverage.



[1]

Figure 3: Basic BIST structure[1]

1.7.1. Test Pattern Generator

There are several different approaches to designing TPGs, however these are the three most referred to in literature[1]:

- Exhaustive testing
- Pseudorandom testing
- Deterministic testing

Exhaustive testing implies generating all possible test patterns for the CUT. For example, using our previous example of the OR gate (Figure 1) as our CUT, we have two inputs into the OR gate, hence four different test patterns. Furthermore, a circuit with 'n' number of inputs has 2^n different test patterns to input in the CUT. As the number of inputs increases in a complex circuit the value for 'n' increases, therefore we will have a large number of different test

patterns. Exhaustive testing entails utilizing all the different (possible) test patterns to test the CUT. However, this approach is counter-effective because testing for all different possible test patterns is time-consuming considering the alternatives.

Pseudorandom testing implies generating 'x' test pattern inputs randomly and generating far fewer test patterns than its counterpart, exhaustive testing. The generation of the test pattern is accomplished using Cellular Automation (CA)[1]. CA encompasses flip-flops interconnected to each other with combinational logic in-between the cells. The combinational logic is responsible for generating the next state of the connected cell. The combinational logic consists of two rules, Rule 90 and Rule 160, which are defined as:

Rule 90: $y_i (t + 1) = y_{i-1} (t)$ XOR $y_{i+1} (t)$

Rule 160: $y_i (t + 1) = y_{i-1} (t)$ XOR $y_i (t)$ XOR $y_{i+1} (t)$

There are other pseudorandom pattern generator schemes; however we utilize the abovementioned scheme for our BIST structure.

The last approach, deterministic testing, fetches test patterns from a pre-stored (ROM) set of test patterns. These test patterns are pre-determined. Deterministic testing implies a full understanding of the CUT and evaluation of what test patterns will yield most faults or, ideally, all of the faults. More importantly, deterministic test patterns are far fewer than exhaustive and pseudorandom test patterns. However deterministic testing is not ideal because in most cases the CUT is not fully realized; hence a pseudorandom testing approach is most effective. Nevertheless, pseudorandom and deterministic testing approaches are realized and implemented in this thesis.

1.7.2. Output Evaluation

The third function of the BIST structure is the output evaluation function. There are several approaches to evaluating fault-output responses with fault-free responses, the following are the most utilized algorithms:

- Transition count
- Signature analysis
- Compaction

Transition count is trivial; it entails the counting of the number of transitions from '0' to '1' or '1' to '0'. For example, the output response '1101101001' has a transition count value of 6. This method does not require the storing of the fault-free response or fault-responses, hence decreasing storage space.

Signature analysis was pioneered by Hewlett-Packard Limited. This approach utilizes data compaction to reduce the data streams by serially inputting the data through an XOR into a shift-register. The XOR gate is XOR-ing the input data with one of the shift-register bits. After all the data inputs have passed through, a special signature is left behind in the shift register and is used to compare its self with other fault-signatures.

Compaction approaches is studied and fulfilled by [10]. The approach studies the CUT and determines the appropriate combinational logic to minimize the number of outputs. Naturally, the compaction approach (which consists of a comparator unit) is utilized with CUTs that contain more than one output; which is the case for most CUTs. This approach is very useful in minimizing the number of outputs, hence decreasing storage space. This approach can cause

some fault-responses to be evaluated as fault-free responses; however zero-aliasing can guarantee (close to 100% in comparison to output evaluation without a comparator) the elimination of error masking or aliasing [11].

1.8. Objectives

In this thesis we present a solution for utilizing hardware as the testing platforms in order to speed up the slow process of fault detection in software. This is achieved by three main solely developed elements and presented in this thesis. The elements are as follows:

- A script to translate BENCH standard files to Very-high-speed integrated circuit Hardware Description Language (VHDL) standard files, while employing fault-injection scheme for every signal.

- A sequential hardware testing algorithm, where every CUT is injected with a fault and tested at each signal for all the CUT's signals sequentially.

- A parallel hardware testing algorithm, where one CUT is instantiated x times to inject a different fault at each instance and tested each instance, simultaneously.

Chapter 2

2. Background

Erstwhile literature has shown fault-injection testing extremely effective when the circuit has been realized that is the classical approach to fault-injection testing. However, another approach is an early stage of fault-injection testing, primarily at the entry level. Several authors demonstrated the advantages of early stage fault-injection.

The authors in [12] examine the usefulness of injecting faults at an early stage. What is proposed is to inject faults in VHDL in comparison to injecting faults after the circuit has been realized. Moreover, their results are promising and illustrate the benefit of early stage fault injection. In [12] we see that this can be done at a very early stage (i.e. directly in the HDL (Hardware Description Language) modules), thence allowing for testing verification during the functional simulation stages. Although functional simulation is very useful to the whole design cycle, these simulations will grow dramatically with the number of injected faults, therefore lacking temporal scalability [13]. Researching further, several authors utilized the FPGA as prototype platform to achieve greater speedups. This approach would provide the testing stage with the physical and real-time inputs and outputs of the digital circuits [14][15]. Therefore, we decided to perform our digital circuit testing using BIST techniques on an FPGA platform.

2.1. Benchmarks

BIST techniques have been verified, implemented, and tested on many benchmarks. These benchmarks were established in the International Symposium on Circuit And Systems (ISCAS) symposium (i.e. c17, c432, etc...). The three categories of benchmarks are combinational, sequential, and core (combinational plus sequential) benchmark circuits. These three benchmarks were established in ISCAS'85, ISCAS'89, and ISCAS'99, respectively, and they will be referred to as ISCAS'85, ISCAS'89, or ISCAS'99 benchmarks. The scope of this thesis covers the former, ISCAS'85 benchmarks (combinational benchmark circuits), such as c17 and c432. Each benchmark has a known number of tests, detected faults, and undetected faults for each test pattern.

2.2. Test Patterns

The test patterns could be generated utilizing one of the following three methods: Deterministic Test Pattern Generator (DTPG), pseudo-Random Test Pattern Generator (RTPG), or Exhaustive Test Pattern Generator (ETPG). DTPG has a known test-set. RTPG could be generated using any pseudo-random techniques, as implemented in [8] and [10]. ETPG implies using the whole truth table's inputs as the test-set.

2.3. Environment

There are two means of implementing test beds for BIST techniques, either using software or hardware. The following two sub-sections (section 2.3.1 and section 2.3.2) describe the software and hardware implementation environments.

2.3.1. Software Environment

The former has been widely and successfully used as a de facto for prototype testing of BIST techniques. The software results obtained for this thesis, and which will be compared to in the subsequent chapters, were obtained by employing the following software simulators: FSIM[18], ATALANTA[19], and HOPE[20]. The software results stem from Dr. Mansour Hanna Assaf and Dr. Assaf's thesis [21]. The thesis introduces circuit testing as an application. We used ATALANTA (fault simulation program developed at Virginia Polytechnic Institute and State University) to generate the fault-free output sequences needed to construct our space compactor circuits and to test the benchmark circuits using reduced test sets accompanied with a random testing session. ATALANTA encompasses both combinational DTPG and ETPG testing methods, however only the former is utilized. FSIM is a fault simulation program utilized to generate pseudorandom test sets necessary to test various combinational circuits; hence FSIM is employed for combinational RTPG testing. And HOPE is another fault simulation program utilized to generate pseudorandom test sets, however HOPE is utilized to test various sequential circuits, not combinational circuit. Therefore, HOPE is outside the scope of this thesis. For each circuit, we determined the number of applied test vectors, the simulation CPU time, and the percentage of fault coverage by employing the above mentioned programs (ATALANTA and FSIM) on a SPARC ULTRA1 SUN workstation.

2.3.2. Hardware Environment

To achieve better performance (i.e. processing time it takes to detect all faults) a hardware system is exploited; more precisely an FPGA (as stated above). Canadian Microelectronic Corporation (CMC) donated the hardware environment. The environment consists of a hardware platform and its appropriate software. The hardware platform is composed of a Virtex-II Multimedia development board designed by Xilinx for multimedia prototyping (Figure 4). The Multimedia board employs a Virtex-II XC2V2000-FF896 as the FPGA. As well a Xilinx MicroBlaze 32-bit soft-core processor is utilized as the processor on-chip. The board is also equipped with five independent banks of 512K x 36-bit 130MHz Zero Bus Turnaround (ZBT) Random Access Memory (RAM) with byte write capabilities.

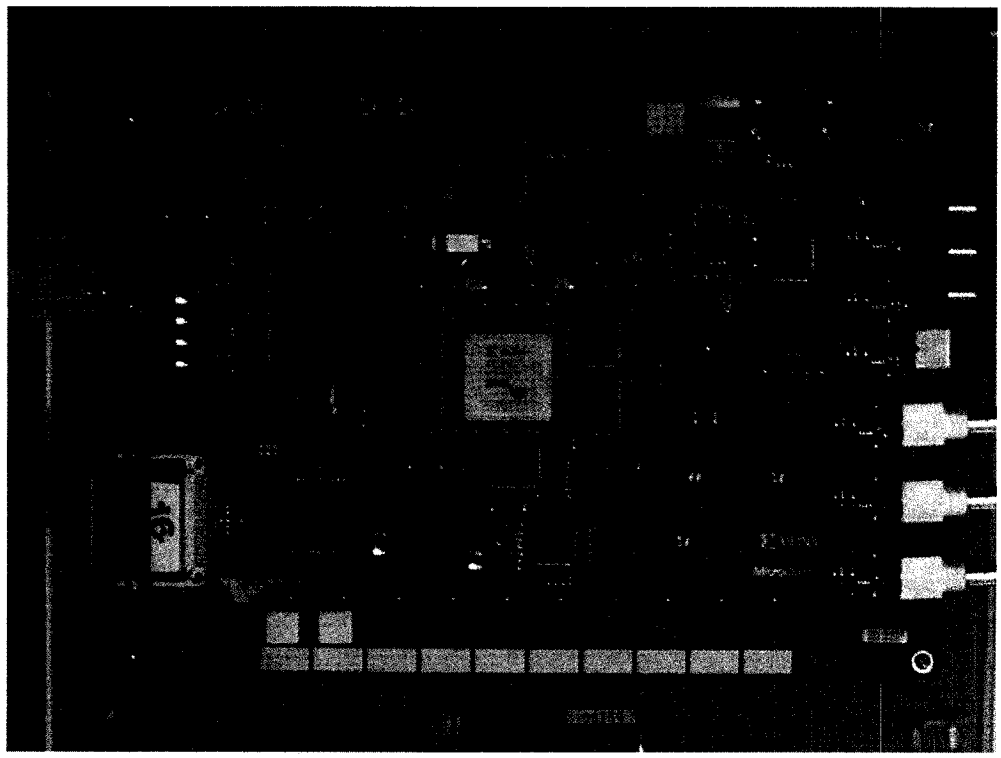


Figure 4: Multimedia board (hardware environment)

Furthermore, the Multimedia board is equipped with a Recommended Standard(RS)-232 port, a keyboard and mouse interface, and an on-board network connection, 10/100 Ethernet, with a Media Access Control (MAC) address. For on-board user inputs/outputs, the board provides 10 pushbuttons, two Dual In-line Package (DIP) switches, and two Light Emitting Diodes (LEDs) for visual feedback.

The Virtex-II FPGA may be programmed via a Xilinx Parallel-4 cable or a Compact Flash (CF) (128 Mbits) card. The latter provides high-speed FPGA configuration, because the CF card utilizes an embedded (on-board) SystemACE [22] controller for configurations. Using either of these methods to program the Virtex-II FPGA requires the implementation, synthesis, and place & route-ing of the system at hand. This is done utilizing the provided Xilinx softwares. The software constructs a BIT file, which would be the configuration file used with either of the abovementioned configuration methods. The softwares are ISE 7.1i and Embedded Design Kit (EDK) 7.1i. The latter is used for embedded system designs and ISE 7.1i is used as a standalone HDL design. The softwares are executed on a Windows XP IBM computer which is, as well, donated by CMC. The IBM computer consists of a 2.8GHz dual-threading Intel Pentium 4 processor, 2GB of RAM, and 160GB of memory. Thanks to CMC, this Personal Computer (PC) allows for dual-threading, whence a system can be simulated and another system can be compiled simultaneously.

2.4. Literature Review

In most hardware applications, hardware performs better then software in execution time hence why several authors have realized hardware

implementations, other than software, which have been developed and tested, such as [23] - [27], to prove hardware's speedup performance.

In [23], it is reported that fault injections are injected at the inputs/outputs of the LUTs (Look Up Tables) and these fault injections are partially reconfigured via a "Boundary Scan" JTAG [22] (Joint Test Action Group) download (hence off-board communication). However this approach limits the testing of each gate because it tests the LUTs which encompass the truth-table of a couple of gates. In this thesis, we resolve this by injecting faults on every gate's inputs/outputs, hence a finer fault injections approach. And secondly, for off-board communication, all fault injection selections are realized on-chip, hence no off-board communication through JTAG, demonstrating a full on-chip on-board BIST system, where all modules of the BIST system are on-chip and are autonomously executed.

In [24] – [27], RTR is utilized to manipulate the fault-injections through small-bit manipulations; one of the two flows for partial reconfiguration [16]. Stuck-at faults are injected into the LUTs by changing the values of the inputs and outputs of the LUTs, utilizing small-bit manipulation, without compromising the executions of the rest of the circuit. The primary downside to this method is the test's accuracy, or lack of. The testing strategy used only injects faults (stuck-at-0 or stuck-at-1) at the inputs or outputs of the CUT, thence restricting the range of discovered hidden and non-hidden faults. As in Figure 5, if only the inputs or outputs of the CUT is injected with faults, the in-between (between the input and output stage) gates and lines are not injected with faults. Hence, in reference to Figure 5, gates 10 and 11 and lines 8, 9, 12, and 13 are not injected with faults. Stuck-at-1 and stuck-at-0 faults are refereed to as s-a-1 and s-a-0, respectively. Keeping in mind Figure 5 as our reference, if we put a s-a-0 at input 2 the output of gate 10 will always be 1, however if line 12 is always s-a-1, by faults which may

or may not cause a failure [1]. There are many faults that would or wouldn't cause similar faults. These faults are stuck-at faults, bridging faults, breaks and transistor stuck-on/-open faults in Complementary Metal-Oxide Semiconductor (CMOS), and delay faults. The scope of thesis only covers the former, stuck-at faults. Nevertheless, all of the pre-mentioned faults are all logical faults as opposed to non-logical faults which include such faults as the malfunction of the clock signal, power failure, and so forth.

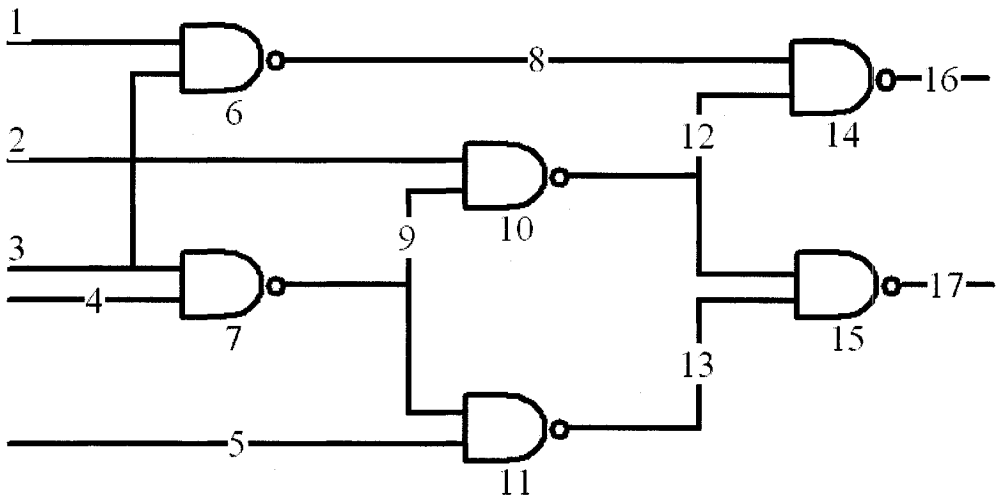


Figure 5: C17 CUT

There is another reason for using other methods of fault injections, other than using LUTs as the inputs/outputs of the CUTs. Figure 5 shows the least complex and smallest circuit of all the ISCAS'85, ISCAS'89, and ISCAS'99 benchmark circuits. Table 1 – Table 3 show how the subsequent circuits grow dramatically. The data in those tables were calculated using the original BENCH files provided from [17]. The circuit name provides the number of gates (which include the inverters) plus the number of lines in the circuit. Clearly the above methods of fault injections [23] – [27] can inject faults into the circuit, having a

colossal drawback to full CUT testing. Moreover, lack of scalability is evident when the circuits are limited to a specific benchmark or a specific CUT, allowing for improvements in their systems.

Table 1: ISCAS'85 benchmark circuits and their respective gate and line counts

ISCAS'85		
Benchmark Circuit Name	Number of Gates*	Number of Lines
C17	6	11
C432	160	272
C499	202	297
C880	383	497
C1355	546	809
C1908	880	1028
C2670	1193	1477
C3540	1669	1871
C5315	2307	3008
C6288	2416	3872
C7552	3512	4040

* These include inverters

Table 2: Eleven least complex ISCAS'89 benchmark circuits and their respective gate, line, and flip-flop counts

ISCAS'89			
Benchmark Circuit Name	Number of Gates*	Number of Lines	Number of Flip-flops
S27	10	17	3
S208	104	104	8
S298	119	179	14
S344	160	184	15
S349	161	188	15
S382	158	224	21
S386	159	227	6
S400	162	238	21
S420	218	202	16
S444	181	263	21

S510	211	299	6

* These include inverters

If we look more closely on [24], we notice a vital missing detail that the authors overlooked. They state that CTR can be realized by "several VHDL codes containing different faults and each of them can be executed separately."[24], however with the right fault injections techniques, like the one in [8]; which uses Fault Injection Multiplexers (FIMs) to inject stuck-at faults into any gate's input or output. A Finite State Machine (FSM) was developed to inject stuck-at faults at every gate's input and output of the CUT, thence a finer discovery of hidden and non-hidden faults with one configuration.

Table 3: Eleven least complex ISCAS'99 benchmark circuits and their respective gate and flip-flop counts

ISCAS'99		
Benchmark Circuit Name	Number of Gates*	Number of Flip-flops
B1	49	5
B2	28	4
B3	160	30
B4	737	66
B5	998	34
B6	56	9
B7	441	49
B8	183	21
B9	170	28
B10	206	17
B11	770	31

* These include inverters

The main objective of our research is to demonstrate the advantages of sequential CTR and parallel CTR and to illustrate the speed-ups associated with digital

circuit testing using BIST for software versus sequential CTR and sequential CTR versus parallel CTR.

Chapter 3

3. System Architecture

The overall system architecture was initially conceived and designed for generic co-processing non-specific application. The system architecture went through a couple of revisions the first of which is on Figure 6[28]. The original architecture was presented in WISES 2004 [28]. That system consists of two high-level components: the Application Flow (AF) and the Reconfiguration Flow (RF), as presented in Figure 6[28]. The former is executed as a software program, with some of its parts being executed on Hardware Blocks (HBs) that are dynamically inserted and removed under RF's control to/from the Reconfigurable Processing Unit (RPU).

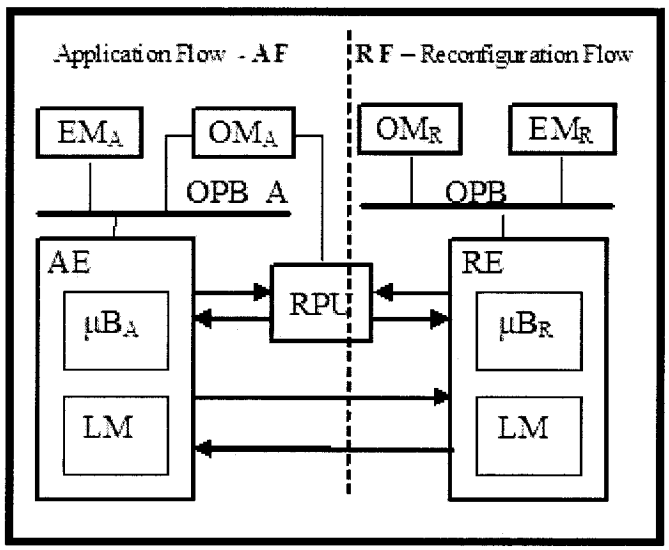


Figure 6: High level system architecture

The hardware/software partitioning of AF is performed by a just-in-time (JIT) compiler, which generates both the AF and RF, as well as ensures proper synchronization between the two flows. The system architecture includes the following components:

- An application element AE consists of a soft IP microprocessor (μB_A) which is used as the execution unit for the main system application;

- A reconfiguration element RE contains a second soft IP microprocessor (μB_R) which is employed as the execution unit for the RTR of the RPU;

- A reconfigurable processing unit (RPU) is used as the execution unit for the tasks to be mapped directly in hardware;

- A shared application on-chip memory (OM_A) stores the user data and instructions for the AF;

- A reconfiguration on-chip memory (OM_R) stores the bitstreams for the different tasks of the RF.

The operation of the application flow (AF) and reconfiguration flow (RF) are shown in Figure 7.

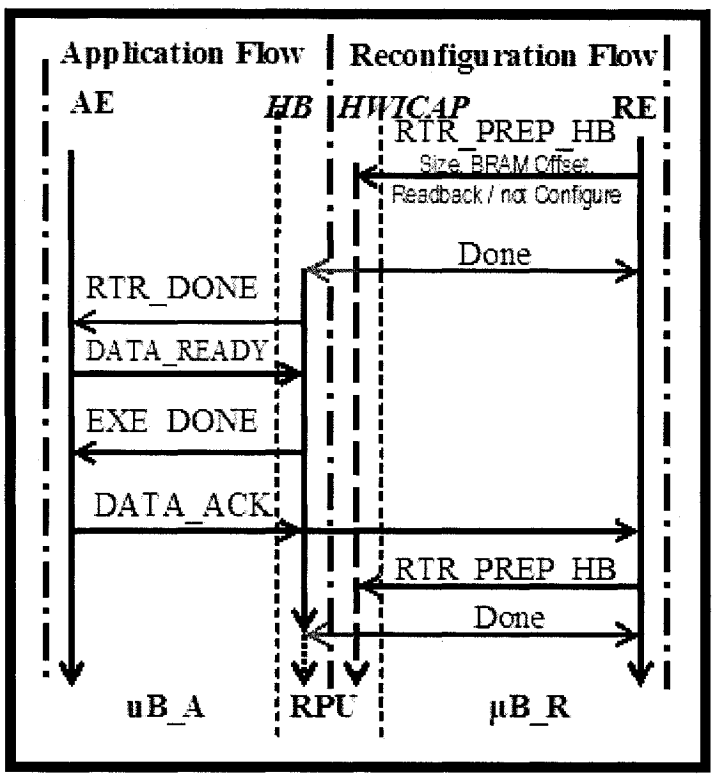


Figure 7: Application and reconfiguration flows of the system

The Internal Configuration Access Port (ICAP) is the FPGA interface that allows for the dynamic partial reconfiguration of the RPU with the appropriate HBs under the control of the reconfiguration flow through the HWICAP controller. While the application flow (AF) runs on the AE, the reconfiguration flow (RF), which runs on the RE, sends the signal RTR_PREP_HB to the ICAP controller (HWICAP) to start the loading of the first HB bitstream (of length "Size", from the address "BRAM Offset" in OMR) onto the RPU. Once that HB is ready within the RPU, the HWICAP sends back a "Done" signal to the RE. The newly placed HB begins execution as soon as it is enabled. Upon completion, the HB sets the flag RTR_DONE to make the application flow aware that it is ready for use. Once the application flow has prepared the data that the HB needs for

computation, it makes the HB aware of this by asserting the DATA_READY flag. The application flow continues to run as long as it does not need the results computed by the HB. The latter asserts EXE_DONE when it completes its task and has prepared the results to be read by the application flow. When the application flow needs these results, it checks the EXE_DONE flag, and blocks if it is not yet set. The application flow gets the results and then asserts DATA_ACK to acknowledge to the HB that it has received the valid data. Once this HB is no longer needed it can be replaced with another one. Also, another HB can be loaded at a different location of RPU as soon as the current uploading process is finished.

The system architecture is composed of three major parts: Just-In-Time (JIT) complier, Run-Time Reconfiguration (RTR), and Reconfigurable Processing Unit (RPU), which are part of the ERACE project by the GEMS Research Lab. The whole project was conceived and realized, from beginning to end, by Dr. Voicu Groza, Rami Abielmona, Mohammad El-Kadri, Arkan Khalaf, and myself. The three components were realized in detail by their respective primes; Mohammad El-Kadri is responsible for the JIT compiler, Arkan Khalaf was responsible for RTR, and I was responsible for the RPU.

Other variations of the architecture were also presented [28], [29], [30], [31], and [32].

3.1. Just-In-Time compiler

The first thesis concentrated on designing an Application Modelling Language (AML) that will create two source codes for each element in the architecture; the Application Element (AE) and the Reconfigurable Element (RE). The former is

meant to operate as the main processor of the whole system, and the latter operates as a co-processor, only executing tasks when initiated by the AE. The flow starts with the user writing their system using the AML, and then the AML transfers the user's requirements to a Just-In-Time (JIT) compiler, which goes about creating the two source codes. The JIT compiler consists of a profiler, partitioner, and synchronizer. The profiler runs through the AML code and searches for "hot spots" (hot spots are areas in the code that will run frequently and/or consume a majority of the execution time). Once the hot spots are located, the partitioner searches the hardware library to see if the found hot spot (usually a function or loop statement) has a replacement implemented in hardware. If that is the case, the partitioner replaces that part of the code with a specific hardware code to be utilized by the synchronizer. The synchronizer will synchronize the code that will be executed on the AE with the code that will be executed on the RE. The AE source code will contain the free-hot spot code and the RE source code will contain the hot spot code. The synchronizer then compiles these sources codes and provides them to the two MicroBlazes employed on each element (AE and RE).

For clarity, let use a simple example of the following three instructions:

1. (5) + (2)

2. Matrix multiplication of a 10 x 10 matrix

3. (result of instruction 2) – (result of instruction 1)

The profile will detect instruction 2 as a hot spot. Then the partitioner will search the hardware library to find an equivalent function to replace the software execution of a matrix multiplication (as we know that matrix multiplication consumes a lot of time). Assuming there is a hardware equivalent function in the

hardware library, we replace instruction 2 with a wait statement. Then we pass the new code to the synchronizer (at this point the only difference in the code is instruction 2 containing a wait statement instead of a matrix multiplication). The synchronizer will output two source codes, one for the AE and another for the RE. The AE source code will contain instruction 1 and 3 with a wait statement before instruction 3 (very important to wait because instruction 3 dependence on instruction 2 results). The RE source code will encompass instruction 2 only. Finally, when the MicroBlazes are powered-on with their respective source codes, the MicroBlazes start executing their first instructions respectively. Since instruction 2 will eventually execute on hardware, therefore instruction 2 should be done executing before instruction 1 is done executing, and if instruction 2 requires longer time, then the wait statement in the AE will wait for the results of instruction 2 then continue to instruction 3. Keep in mind, this is a very simple example, however it conveys the value of the JIT complier into the whole project. Henceforth, in summary this system is composed of two high-level components, namely the application flow (AE) and the reconfiguration flow (RE). These two flows will be generated by a **J**ust-**I**n-**T**ime (JIT) compiler of the source code which will extract the flows from the latter as well as ensure synchronization between them. This JIT compiler is outside the scope of this thesis.

3.2. Run-Time Reconfiguration

Run-Time Reconfiguration allows part of the FPGA to be reconfigured while the other parts are functioning uninterruptedly. The RTR thesis allows the project greater flexibility by allowing the execution of different hardware functions (hardware function will be referred to as Hardware Block (HB)) at different time slots. Suppose the space allocated for an HB is only big enough for one

instruction of the size of the matrix multiplication. Let's introduce a 4th instruction, which is a matrix division. Since there is no space for the 4th instruction, given that the 2nd instruction is occupying the space allocated for the HBs, we cannot execute the 4th instruction in hardware. However, with RTR, once an HB has expired its lifespan (once the results for instruction 2 are attained), the new HB (4th instruction) can be reconfigured on the FPGA while the rest of the FPGA is operating (this operation is usually termed "on-the-fly" configuration). This RTR is outside the scope of this thesis.

3.3. Reconfigurable Processing Unit

The above architecture is useful as a co-processor design, benefiting in speedups and power conservation. However our BIST application does not require a co-processor and will be executed as a stand-alone processing unit. Henceforth the proposed final architecture is a modification of the original architecture. These modifications were the elimination of AE and OM_R blocks and the removal of the MAB because the RcP can access the OM_A via the ComBus. As well RcP was renamed to RPU because this block's functions are no longer executed as a co-processor but as a hardware accelerator and OM_A was just rename OM. Figure 8 illustrates the modified application specific architecture. To revive the idea of our application, the testing of digital circuits would be executed in the RPU and the μB_R would be our soft-core processor to maintain statistics on bus utilization.

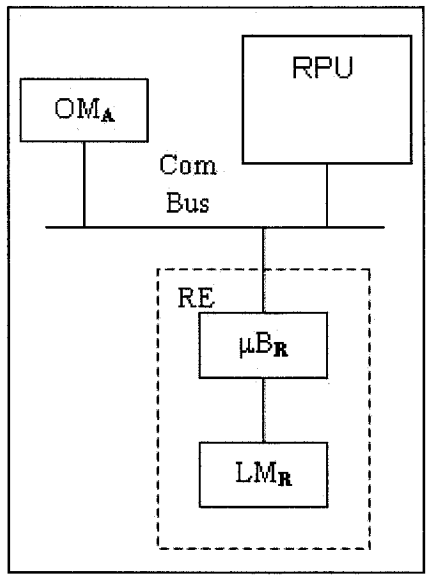


Figure 8: Modified application specific system architecture

Elaborating further, the above architecture figures are high-level block diagrams and they were designed without significant parallelism and multi-tasking in mind. While Figure 9 and Figure 10 are more detailed high-level block diagrams which have more tasks utilizing the buses to illustrate the multi-tasking and parallelism depth. Furthermore, Figure 9 and Figure 10 were realized from Figure 6 and Figure 8 respectively. Also, it should be noted that the notation μB_R refers to MicroBlaze Reconfiguration block and the notation OM_A refers to On-chip Memory Application.

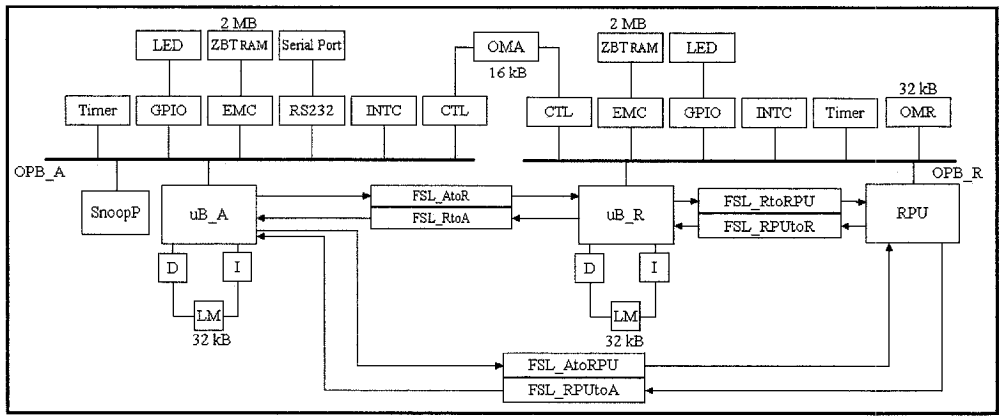


Figure 9: Detailed high-level system architecture

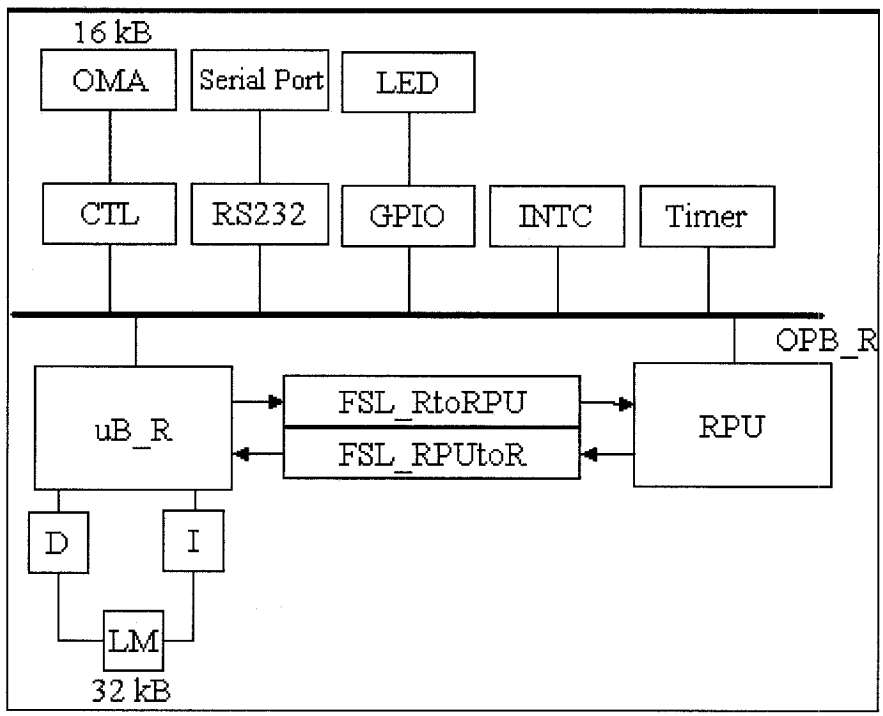


Figure 10: Modified and detailed application specific system architecture

Hitherto we haven't explained our final architecture for our specific application. We have only demonstrated the thought process to approach our final

architecture. Accordingly, the proposed final architecture, as shown on Figure 10, consists of an RPU attached on the OPB_R. The RPU's own design went through a couple of stages. The static design was designed as a static co-processor (hardware accelerator), while the dynamic design utilizes RTR functionality in order to respond to the changing environment inputs. The RPU allows for the simultaneous execution of multiple hardware units, by exploiting a JIT compiler in order to maintain hardware and software flow synchronization. The JIT compiler's description and RTR is outside the scope of this thesis.

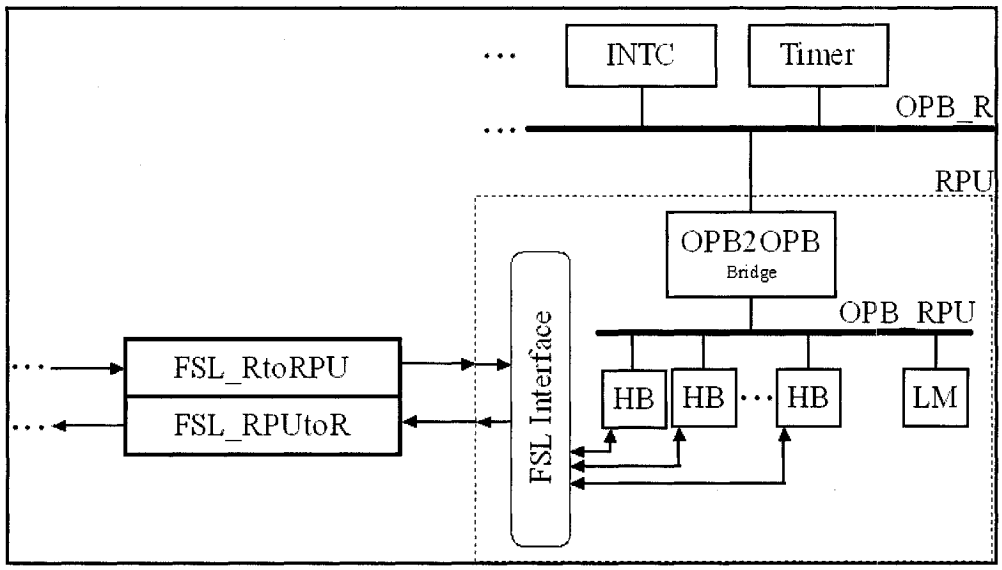


Figure 11: The RPU high-level block diagram

The following is a brief description of the components in Figure 10 and Figure 11:

OPB_R: This is the **O**n-board **P**eripheral **B**us[22] utilized by the components for communication.

uB_R: This is the MicroBlaze utilized as a soft-core processor. The μB_R will aid in gathering statistics on bus utilization and screen outputs of internal registers. The screen outputs are sent via the serial port.

LM: This is the **L**ocal **M**emory which houses the μB_R source code. The LM connects to the μB_R via a **D**ata (**D**) and an **I**nstruction (**I**) bus. The size of the LM is 32 KB.

RPU: This is the Reconfigurable Processing Unit which is where the functional hardware is executed. The RPU encompassed different HBs, and for our application these HBs are CUTs (c17, c432, etc…).

FSL: This is the **F**ast **S**implex **L**ink (FSL) to and from μB_R and RPU. These FSLs allow the MicroBlazes to communicate to the appropriate HBs, within the RPU, directly.

OMA: This is the **O**n-chip **M**emory. The OM_A will store the user data and the results of the HBs.

LED, **INTC**, and **Timer**: These are leds, interrupt controller, and timer respectively. These are other tasks that are utilizing the bus as well.

Serial Port: This is just used as a tool by the μB_R to output data to the user.

OPB2OPB Bridge: This is a bridging component from OPB_R to OPB_RPU to allow for the MicroBlaze to communicate through OPB_R to the HBs. As well we propose the addition of **L**ocal **M**emory (LM) to OPB_RPU to allow inter-communication between HBs.

That is our system architecture used to test digital circuits. Each HB encompasses the CUT plus the utilized BIST technique. There are two different

strategies that are employed, **Sequential CTR** (**C**ompile-**T**ime **R**econfiguration) and **Parallel CTR**.

3.4. Sequential Compile-Time Reconfiguration

This strategy is a Sequential CTR realization of the fault-model (Figure 12), where the CUT and compressors [21] are synthesized and mapped to the target device, with **FIM**s (discussed below in section 3.8.1) built into the CUT. The compressor is a module which executes a method of reducing the number of bits in the original circuit response during testing in which no or minimal information is lost, so the original output sequence can be fully regenerated from the compressed sequence[33].

This allows us to sequentially perform test verification on the circuit by applying the test patterns and recording the circuit's behaviour for each applied test pattern and injected fault. This will provide us with our sequential hardware benchmark results.

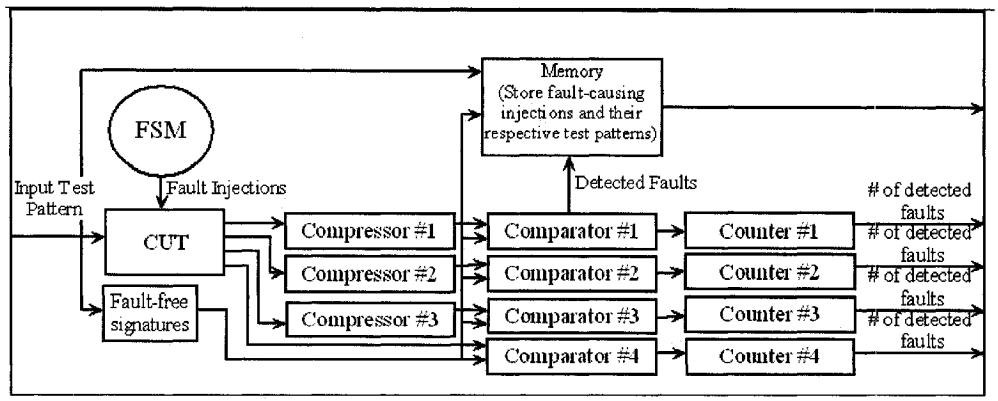


Figure 12: Sequential CTR High-Level Block Diagram

In Figure 12, note that the use of three compressors (Compressor #1 – #3) is to identify the most efficient compressor and the compressor with the least amount of information lost and we use a comparator (Comparator #1 – #3) for each compressor and another comparator (comparator #4) to test the effect of not utilizing a compressor.

Figure 12 shows the HB architecture and all the inner modules. The BIST technique is encompassed within the CUT and FSM. We introduce a stuck-at-0 and a stuck-at-1 for each line in the circuit; hence in Figure 5 (C17 benchmark circuit) we would introduce 22 stuck-at faults (11 stuck-at-0 and 11 stuck-at-1 faults). These stuck at faults are injected utilizing FIMs technique. The hardware fault injection technique is imperative in order to iteratively inject faults to every mutually-exclusive wire, and to test both the stuck-at-0 and stuck-a-1 faults. For that matter, a plan was formulated that is realized from[8] and will be elaborated on in section 3.8.1.

The flow of FSM for executing the CUTs is shown in Figure 13. The flow starts of by initializing the values of the FSM, and then injecting a stuck-at on one of the lines, and then accepts the test pattern input. From the test pattern input we

know the real output (i.e. the CUT's output without any fault injections) and we have the fault injected CUT output. We then use 3 different compressors to see if any of them will make a significant difference for future research. Each compressor is designed differently and respectively to the CUT. Once at the comparator stage, we detected if there are any discrepancies between the real outputs and the fault injected CUT's outputs. If a fault is detected we increment a counter for the respective compressor, as well we keep track of the fault causing injections and their respective test patterns. These values are read by the μB_R and then displayed for the user. Then we repeat the above steps for a different input until all the inputs have been executed, and then we inject a stuck-at on a different line. We are done testing once we run every permutation of test patterns and fault injections.

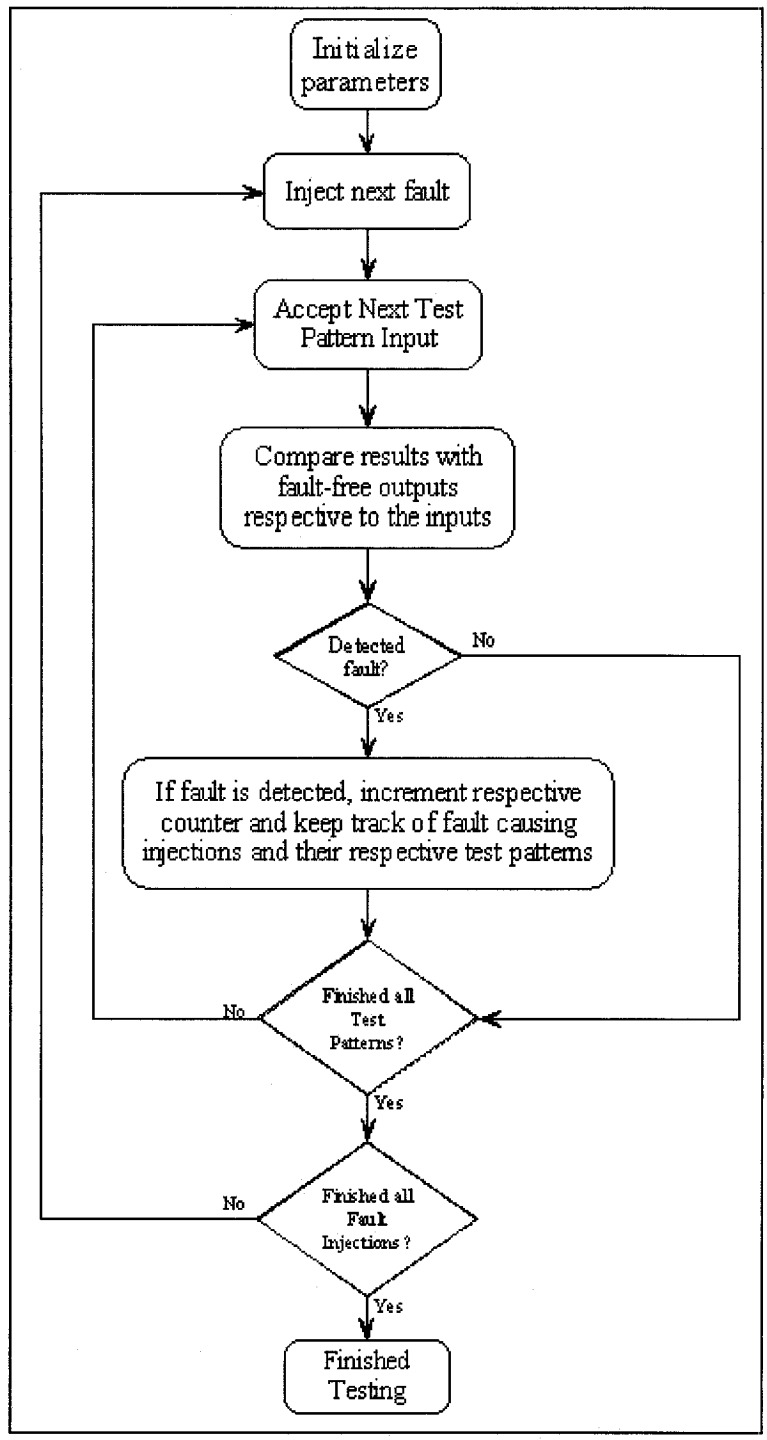


Figure 13: Sequential CTR FSM flow

3.5. Parallel Compile-Time Reconfiguration

This strategy is a partially-Parallel CTR realization of the fault-model, where n * m * 2 CUTs are synthesized and mapped to the target device, with one particular fault-injection applied to each CUT. Hence, the test patterns are applied to all the CUTs simultaneously, providing us with the circuit's behaviour for each applied test pattern only. Note that in a fully-parallel strategy, n * m * l * 2 CUTs are synthesized and mapped, where,

n is the number of wires in the CUT;

m is the number of test patterns used;

l is the number of available test patterns; and

2 is the number of stuck-at faults to inject.

Figure 14 demonstrates how the high level block diagram is represented. Moreover, if you compare parallel CTR with sequential CTR you will notice the differences are the CUTs are synthesized with one faults already injected.

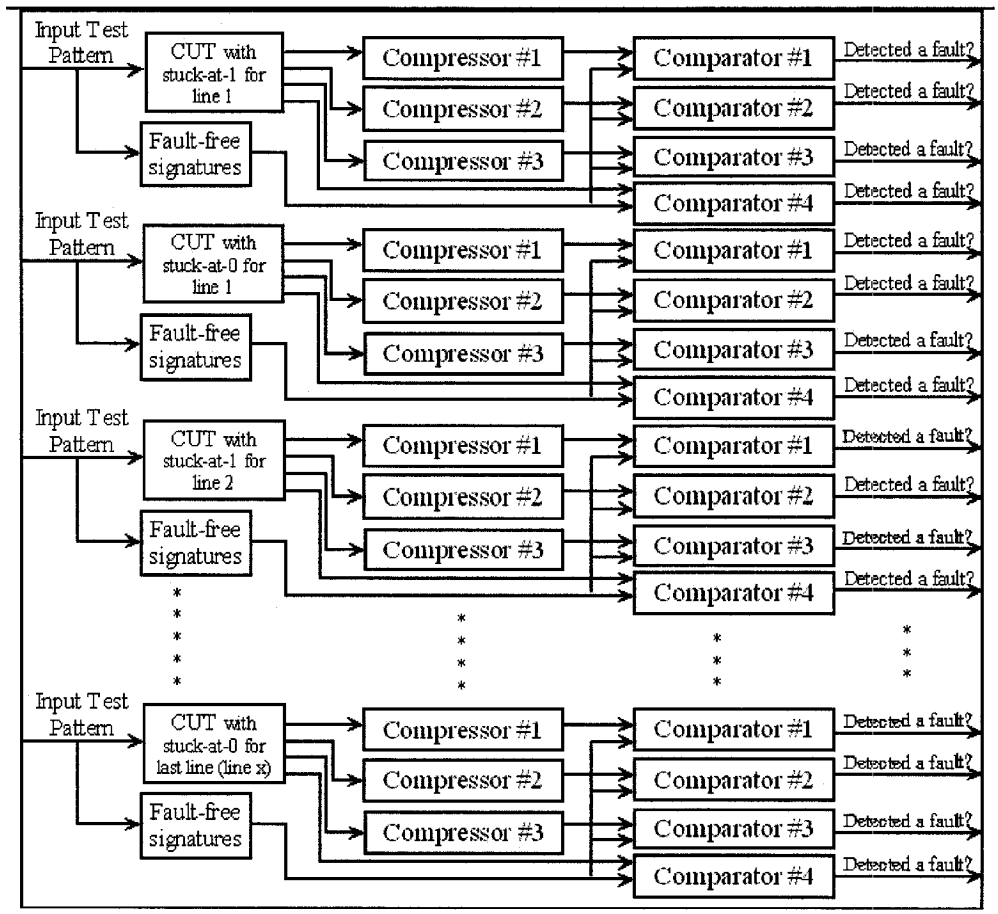


Figure 14: Parallel CTR High-Level Block Diagram

Conversely, in a partially-parallel strategy, only n * m * 2 CUTs are required. Hence, if realizing a fully-parallel strategy, one clock cycle is required to generate the circuit's behaviour for all applications of test patters and injected faults. This strategy was not utilized due to the unrealistic abuse of logical resources, as well as the fact that all the test patterns are not known at compile-time, specifically in the pseudo-random test pattern generation stage. Therefore, partially-parallel CTR strategy is referred to as parallel CTR strategy. This strategy is similar, in implementation, to the sequential CTR strategy, however, the FSM (Figure 15)

does not inject any faults into the CUT, as they are already synthesized into the circuit. This will provide us with our parallel hardware benchmark results.

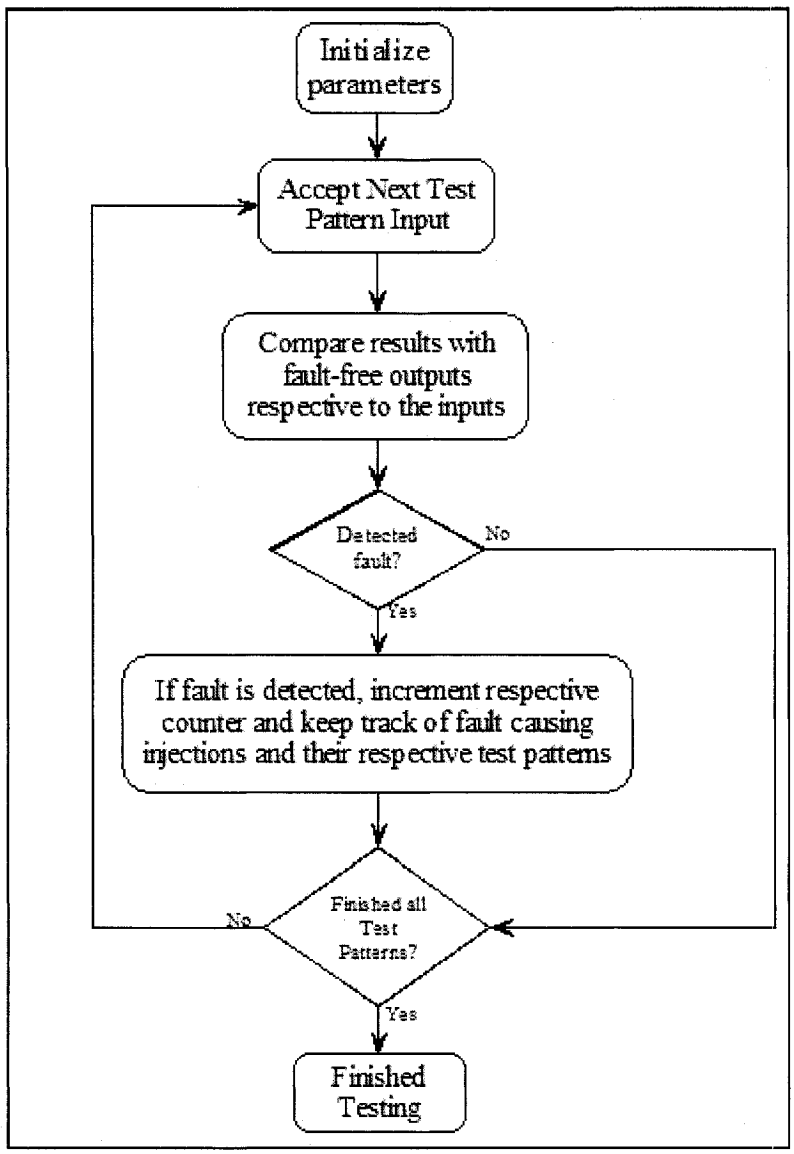


Figure 15: Parallel CTR FSM flow

3.6. Pseudo-Random Number Generator

A hardware Random Number Generator (RNG) was developed by Rami Abielmona [34]. Having understood the full implementation of the RNG (utilizing rule 90 and rule 160) we have decided to utilize the RNG for our RTPG.

3.7. Script to convert BENCH to VHDL

A script was designed and implanted using Python 2.4 to create VHDL files by parsing through BENCH files. BENCH files are the standard format of files for benchmark circuit. The following is an example of the C17 BENCH file:

```
# c17
# 5 inputs
# 2 outputs
# 0 inverter
# 6 gates ( 6 NANDs )

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = NAND(3, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

The inputs and outputs include the wire number with-in the parenthesis. The last block include the new wire number and what Boolean function make-up that wire. Therefore, wire 10 is a NAND gate of wire 1 and wire 3.

The first challenge is that the number sequence used is not sequential. So, that was the first thing the parser executes. The result being this:

```
# c17
# 5 inputs
# 2 outputs
# 0 inverter
# 6 gates ( 6 NANDs )

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(10)
OUTPUT(11)

6 = NAND(1, 3)
7 = NAND(3, 4)
8 = NAND(2, 7)
9 = NAND(7, 5)
10 = NAND(6, 8)
11 = NAND(8, 9)
```

As shown, the wire numbers are sequenced. This was important, because we did not want to discard any signal declarations in the VHDL code, therefore only having to declare 1 to 11 wires (signals).

The next step is trivial if you do not have to take into consideration that a multiplexer has to be inserted at every wire (signal). To solve this challenge we

split every wire into two wires, one wire is pre-multiplexer and the other wire is post-multiplexer. By observing the above modified BENCH file code it's trivial that the input signals tie to the pre-multiplexed signals and the output signals tie to the post-multiplexed signals. The last block, utilizes pre-multiplexed signals between the parenthesis and the output signal is tied to post-multiplexed signals. By slightly increasing the complexity of our code we can now use the GENERATE function in VHDL to generate 11 multiplexers all of which have the pre-multiplexed signals tied to the post-multiplexed signals. Here is the VHDL code, the parser create, for converting C17 BENCH file to VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity c17_cut is
  generic
  (
    I_DWIDTH:    integer := 5;
    O_DWIDTH:    integer := 2;
    NUM_WIRE:    integer := 11
  );
  port
  (
    i_data      : in std_logic_vector(0 to I_DWIDTH-1);
    i_select    : in std_logic_vector(0 to NUM_WIRE*2); -- You need from 0 to 22 selects inputs to pick the
value for a specific mux.
    o_data      : out std_logic_vector(0 to O_DWIDTH-1)
  );
end c17_cut;

architecture rtl of c17_cut is
  signal wire_in2_mux : std_logic_vector(0 to NUM_WIRE);
  signal wire_out_mux : std_logic_vector(0 to NUM_WIRE);
  type MUX_INPUT_TYPE is array(0 to NUM_WIRE) of std_logic_vector(0 to 1); -- This will make a
double array of length 11 with two values for each select input.
```

```
  signal mux_select : MUX_INPUT_TYPE;

begin
  -- input values
  wire_in2_mux(1) <= i_data(0);
  wire_in2_mux(2) <= i_data(1);
  wire_in2_mux(3) <= i_data(2);
  wire_in2_mux(4) <= i_data(3);
  wire_in2_mux(5) <= i_data(4);

  -- output values
  o_data(0) <= wire_out_mux(10);
  o_data(1) <= wire_out_mux(11);

  -- internal values
  wire_in2_mux(6) <= not(wire_out_mux(1) and wire_out_mux( 3));
  wire_in2_mux(7) <= not(wire_out_mux(3) and wire_out_mux( 4));
  wire_in2_mux(8) <= not(wire_out_mux(2) and wire_out_mux( 7));
  wire_in2_mux(9) <= not(wire_out_mux(7) and wire_out_mux( 5));
  wire_in2_mux(10) <= not(wire_out_mux(6) and wire_out_mux( 8));
  wire_in2_mux(11) <= not(wire_out_mux(8) and wire_out_mux( 9));

  -- Try to generate a function to convert 22 inputs into 11 inputs with two values each.  Hence,
  -- making a double array from a single array.
  convert : for j in 1 to NUM_WIRE generate
    mux_select(j)(0) <= i_select(2*j-1);
    mux_select(j)(1) <= i_select(2*j);
  end generate convert;

  multiplexers : for i in 1 to NUM_WIRE generate
    wire_out_mux(i) <=  '0' when (mux_select(i)="10") else
                '1' when (mux_select(i)="01") else
                wire_in2_mux(i);
  end generate multiplexers;

end rtl;
```

3.8. Fault-Injection Techniques

In this work, two fault-injection techniques could be utilized for hardware designs. The first involves fault-injection multiplexers, and is used in the sequential CTR strategy, while the second fault-injection technique involves fault-injection LUT, and could be used in parallel CTR and other advanced hardware configuration (RTR).

3.8.1. Fault-Injection Multiplexer

The hardware fault injection technique is imperative in order to iteratively inject faults to every mutually-exclusive wire, and to test both the stuck-at-0 and stuck-at-1 faults. As we can see in the figure (Figure 16), every mutually-exclusive wire now has a FIM introduced within it, which allows us to either run the wire as is, or inject stuck-at-0 or stuck-at-1 faults. If the value of the select signals of any multiplexer are at "00", then we will run the wire as is, while if the values are at "01", then we will inject a stuck-at-1 fault indicated by the logical 1 value coming into the multiplexer, and if the values are at "10", then we will inject a stuck-at-0 fault indicated by the logical 0 value coming into the multiplexer. Finally, if the values are at "11", then we will again assume normal operation of the wire.

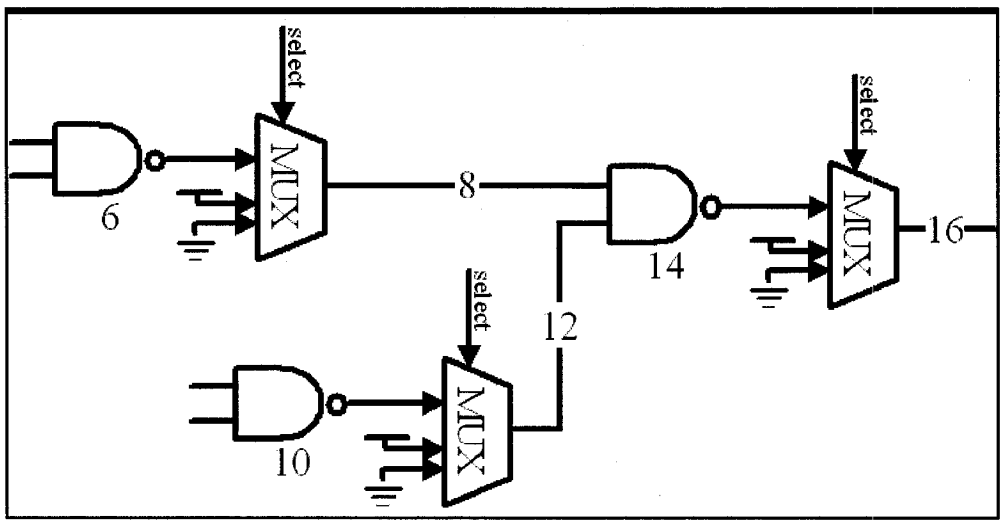


Figure 16: The FIMs scheme in hardware (this is part of the C17 CUT)

Because of the design of the FIMs and the way the VHDL code has been generated it is possible to receive a vector input to inject the fault with only one of the bits set to '1'. Hence you would have '1000000000000000' vector as the fault injection input vector. The vector will not to inject a fault into the circuit because the first bit is not used, this will get the user an opportunity to retrieve the true output of the circuit. However, '0100000000000000' injects a s-a-0 on line 1, then to inject a s-a-1 on line 1 all we have to do is shift the vector right by one, therefore '0010000000000000'. If you keep shifting right, that will allow the circuit to be injected with every possible fault injections, henceforth the parser made complicated things to eventually make them simply during execution and generalization of the parser allows the parser to create any combinational circuit BENCH file into a VHDL file with multiplexers that are easily accessible.

3.8.2. Fault-Injection Look Up Table

The hardware fault injection LUT is essential in order to inject faults to circuits at the post-synthesis stage. Transpiring in the synthesis stage, all the logical circuits are placed into LUTs that represent the circuits' input and output behaviour, hence not permitting the user access to inject faults to every mutually-exclusive wire (this point was discussed in detail in the Background section), however it allows the user to inject faults at the post-synthesis stage. Conversely, unlike sequential CTR, parallel CTR doesn't require FIMs nor fault-injection LUTs as its strategy, we can use the latter however for simplicity we simply inject faults early in the design stage, at the HDL stage, pre-synthesis.

Chapter 4

4. Results, show-offs, and trade-offs

Software, sequential hardware, and parallel hardware circuit testing are compared in this chapter. The focus is to determine the trade/show-offs associated with the fundamentals of the three approaches: software costs increase gradually with circuit complexity, test length and fault list size. Sequential hardware has a significant initial cost, associated with FPGA synthesis and implementation; however the execution speed is the trade-off. Parallel hardware has the same costs as its predecessor (sequential hardware), plus the costs of memory because of its nature of duplication, then again the execution speed is greater than its forerunner and much greater than software.

In order to compare the three approaches, two ISCAS'85 circuits, c17 and c432, and two recognized circuits, ec13 (full-adder) and ec37, have been used as test vehicles.

Software simulation has been carried out using a widely used fault simulation tools, ATALANTA and FSIM. Furthermore, hardware testing was carried out using the Xilinx Multimedia board encompassing a Xilinx Virtex-II 2000 FPGA, operating at a frequency of 50 MHz, together with Xilinx Platform Studio (EDK) v. 7.1i.

The following are performance measures that will be extracted from our hardware circuit testing:

i. Speed of computation of each BIST implementation

ii. Number of faults detected versus number of faults injected (i.e. fault coverage and number of faults undetected)

iii. Number of test patterns versus total test time

iv. Memory usage

v. Bus utilization

From the software aspects, only the first four performance measures are available.

We executed four different digital circuits, c17, c432, ec13, and ec37. The former two circuits (c17 and c432), as aforementioned, are benchmark circuits. The latter two circuits (ec13 and ec37) are circuits used in literature to give a variety of testing features, since the gap between c17 and c432 is large (with respect to number of gates and number of lines, see Table 1). The ec13 and ec37 circuits do not retain all of the abovementioned software performance of measures. Nevertheless, these circuits illustrate the hardware circuits' modularity in executing different circuits.

4.1. C17 benchmark circuit

The experimental results obtained from software testing of a deterministic and a pseudorandom Test Pattern Generations (TPG) of the C17 circuit are as follows:

Deterministic testing (according to ATALANTA)

	Input (I)	Output (O)	
Test 1:	01111	00	8 faults detected

Test 2:	11010	11	8 faults detected
Test 3:	10000	00	3 faults detected
Test 4:	10101	11	3 faults detected

Fault coverage:	100%
Number of test patterns applied:	4
Number of detected faults:	22
Memory used:	2433 Kbytes
CPU time:	0.017 seconds

From these results we can see that the deterministic test patterns were used as inputs (i.e. the test patterns were provided before testing commenced) for the C17 circuit and next to the test pattern inputs are their respective fault-free outputs. With every test pattern input, the program (ATALANTA and FSIM have the same output format) injects a fault on every signal and counts the number of faults detected from those injections (i.e. we keep the input fixed and we inject all the faults and we detect for each injection to detect a fault). As we can see in the C17 circuit, test 1 detects 8 faults out of 22 fault injections, test 2 detects 8 faults out of 22 fault injections, test 3 detects 3 faults out of 22 fault injections, and test 4 detects 3 faults out of 22 fault injections. If the total number of detected faults from all the tests (i.e. 8 + 8 + 3 + 3) is equal to the total number of fault injections for each test (i.e. 22), then the fault coverage is a 100%. As mentioned in the previous chapter, ATALANTA and FSIM were executed on a SUN SPARC unit, hence CPU time and memory used is referring to the execution time needed and memory used to execute all of the tests. Lastly, the total number of test patterns applied is 4 as compared to 32 for the pseudorandom testing. For deterministic and pseudorandom testing, the total number of test patterns applied is predetermined.

In this chapter, all of the result formats for ATANLANTA and FSIM programs will have the same formats as described in previous paragraph. Note that 'I' refers to Input and 'O' refers to Output.

Pseudorandom testing (according to FSIM)

	I	O	
Test 1:	00000	00	5 faults detected

Test 2: 00110 00 1 faults detected
Test 3: 01010 11 8 faults detected
Test 4: 01101 11 1 faults detected
Test 5: 00011 01 1 faults detected
Test 6: 00110 00 0 faults detected
Test 7: 11101 11 0 faults detected
Test 8: 00000 00 0 faults detected
Test 9: 10110 10 2 faults detected
Test 10: 00001 01 0 faults detected
Test 11: 10110 10 0 faults detected
Test 12: 10110 10 0 faults detected
Test 13: 10011 01 1 faults detected
Test 14: 11000 11 0 faults detected
Test 15: 00111 00 2 faults detected
Test 16: 00000 00 0 faults detected
Test 17: 11001 11 0 faults detected
Test 18: 10111 10 0 faults detected
Test 19: 10100 10 0 faults detected
Test 20: 01111 00 1 faults detected
Test 21: 01010 11 0 faults detected
Test 22: 11111 10 0 faults detected
Test 23: 10101 11 0 faults detected
Test 24: 11011 11 0 faults detected
Test 25: 01001 11 0 faults detected
Test 26: 00101 01 0 faults detected
Test 27: 00101 01 0 faults detected
Test 28: 00100 00 0 faults detected
Test 29: 11101 11 0 faults detected
Test 30: 10010 00 0 faults detected
Test 31: 00110 00 0 faults detected
Test 32: 11010 11 0 faults detected

Fault coverage:	100%
Number of test patterns applied:	32
Number of detected faults:	22
Memory used:	397 Kbytes
CPU time:	0.017 seconds

The above results were accomplished using and software simulators ATLANTA and FSIM, both of which don't provide the fault occurrence (i.e. at which line did

the faults occur). However, in our design we will extract the exact line that caused the faults. Moreover, our design will reveal if the occurring fault was a stuck-at-0 or a stuck-at-1 fault.

The experimental results obtained from hardware sequential testing of a deterministic and a pseudorandom TPG of the C17 are as follows:

Deterministic sequential testing

	I	O	
Test 1:	01111	00	9 faults detected
Test 2:	11010	11	6 faults detected
Test 3:	10000	00	2 faults detected
Test 4:	10101	11	5 faults detected

Fault coverage:	100%
Number of test patterns applied:	4
Number of detected faults:	22
Memory used:	32 Kbytes
CC:	88 Clocks
@ 50 MHz	1.76 μs

Localization:
Input: 01111 Circuit Output: 10 True Output: 00 Mux-1 Stuck-at-1
Input: 01111 Circuit Output: 11 True Output: 00 Mux-3 Stuck-at-0
Input: 01111 Circuit Output: 11 True Output: 00 Mux-4 Stuck-at-0
Input: 01111 Circuit Output: 10 True Output: 00 Mux-6 Stuck-at-0
Input: 01111 Circuit Output: 11 True Output: 00 Mux-7 Stuck-at-1
Input: 01111 Circuit Output: 11 True Output: 00 Mux-8 Stuck-at-0
Input: 01111 Circuit Output: 01 True Output: 00 Mux-9 Stuck-at-0
Input: 01111 Circuit Output: 10 True Output: 00 Mux-10 Stuck-at-1
Input: 01111 Circuit Output: 01 True Output: 00 Mux-11 Stuck-at-1
Input: 11010 Circuit Output: 00 True Output: 11 Mux-2 Stuck-at-0
Input: 11010 Circuit Output: 10 True Output: 11 Mux-3 Stuck-at-1
Input: 11010 Circuit Output: 00 True Output: 11 Mux-7 Stuck-at-0
Input: 11010 Circuit Output: 00 True Output: 11 Mux-8 Stuck-at-1
Input: 11010 Circuit Output: 01 True Output: 11 Mux-10 Stuck-at-0
Input: 11010 Circuit Output: 10 True Output: 11 Mux-11 Stuck-at-0

Input: 10000 Circuit Output: 11 True Output: 00 Mux-2 Stuck-at-1
Input: 10000 Circuit Output: 01 True Output: 00 Mux-5 Stuck-at-1
Input: 10101 Circuit Output: 01 True Output: 11 Mux-1 Stuck-at-0
Input: 10101 Circuit Output: 10 True Output: 11 Mux-4 Stuck-at-1
Input: 10101 Circuit Output: 10 True Output: 11 Mux-5 Stuck-at-0
Input: 10101 Circuit Output: 01 True Output: 11 Mux-6 Stuck-at-1
Input: 10101 Circuit Output: 10 True Output: 11 Mux-9 Stuck-at-1

As we can see the above output format only differs from ATALANTA and FSIM by the addition of a localization table. The localization table localizes the fault, hence it localizes the signal (wire) the fault occurred at and if a s-a-0 (stuck-at-0) or s-a-1 (stuck-at-1) caused the fault to be detected. This table is important for comparison with other literature and vice-verse. Since hardware is best measured with Clock Cycles (CC), we have employed a clock counter to keep track of the number of CC needed to complete testing. The CC and its equivalence to time with respect to the clock speed (50MHz) are presented. The same output format is presented for all hardware testing (sequential CTR or parallel CTR).

Pseudorandom sequential testing

	I	O	
Test 1:	11010	11	6 faults detected
Test 2:	11101	11	1 faults detected
Test 3:	00100	00	8 faults detected
Test 4:	00111	00	3 faults detected
Test 5:	00001	01	2 faults detected
Test 6:	11110	10	2 faults detected

Fault coverage:	100%
Number of test patterns applied:	63
Number of detected faults:	22
Memory used:	32 Kbytes

CC:	1386 Clocks
@ 50 MHz	27.72 μs

Localization:
Input: 11010 Circuit Output: 11 True Output: 00 Mux-2 Stuck-at-0
Input: 11010 Circuit Output: 11 True Output: 10 Mux-3 Stuck-at-1
Input: 11010 Circuit Output: 11 True Output: 00 Mux-7 Stuck-at-0
Input: 11010 Circuit Output: 11 True Output: 00 Mux-8 Stuck-at-1
Input: 11010 Circuit Output: 11 True Output: 01 Mux-10 Stuck-at-0
Input: 11010 Circuit Output: 11 True Output: 10 Mux-11 Stuck-at-0
Input: 11101 Circuit Output: 11 True Output: 10 Mux-4 Stuck-at-1
Input: 00100 Circuit Output: 00 True Output: 10 Mux-1 Stuck-at-1
Input: 00100 Circuit Output: 00 True Output: 11 Mux-2 Stuck-at-1
Input: 00100 Circuit Output: 00 True Output: 01 Mux-5 Stuck-at-1
Input: 00100 Circuit Output: 00 True Output: 10 Mux-6 Stuck-at-0
Input: 00100 Circuit Output: 00 True Output: 11 Mux-8 Stuck-at-0
Input: 00100 Circuit Output: 00 True Output: 01 Mux-9 Stuck-at-0
Input: 00100 Circuit Output: 00 True Output: 10 Mux-10 Stuck-at-1
Input: 00100 Circuit Output: 00 True Output: 01 Mux-11 Stuck-at-1
Input: 00111 Circuit Output: 00 True Output: 01 Mux-3 Stuck-at-0
Input: 00111 Circuit Output: 00 True Output: 01 Mux-4 Stuck-at-0
Input: 00111 Circuit Output: 00 True Output: 01 Mux-7 Stuck-at-1
Input: 00001 Circuit Output: 01 True Output: 00 Mux-5 Stuck-at-0
Input: 00001 Circuit Output: 01 True Output: 00 Mux-9 Stuck-at-1
Input: 11110 Circuit Output: 10 True Output: 00 Mux-1 Stuck-at-0
Input: 11110 Circuit Output: 10 True Output: 00 Mux-6 Stuck-at-1

Since we are only testing combinational circuit we can calculate the expected execution time to test one CUT. To compare our hardware expected results with software results, we have executed the c17 using the ATALANTA simulator, with 4 test inputs (deterministic testing) (i.e. 01111, 11010, 10000, and 10101). The simulator needed a total time of 17 ms to finish execution. Our hardware needs 88 Clock Cycles (CC) for deterministic testing, which translates to 1.76 μs, with a 50 MHz clock; transforming into a speedup of 1×10^4. And our hardware needs 1386 CC for random testing, which translates to 27.72 μs, with a 50 MHz clock; transforming into a speedup of 6×10^2.

Nevertheless, these speedups can even be improved with a parallel hardware implementation at the cost of memory.

The experimental results obtained from hardware parallel testing of a deterministic and a pseudorandom TPG of the C17 are as follows:

Deterministic parallel testing

	I	O	
Test 1:	01111	00	9 faults detected
Test 2:	11010	11	6 faults detected
Test 3:	10000	00	2 faults detected
Test 4:	10101	11	5 faults detected

Fault coverage:	100%
Number of test patterns applied:	4
Number of detected faults:	22
Memory used:	32 Kbytes
CC:	4 Clocks
@ 50 MHz	0.08 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

Pseudorandom parallel testing

	I	O	
Test 1:	11010	11	6 faults detected
Test 2:	11101	11	1 faults detected
Test 3:	00100	00	8 faults detected
Test 4:	00111	00	3 faults detected
Test 5:	00001	01	2 faults detected
Test 6:	11110	10	2 faults detected

Fault coverage:	100%
Number of test patterns applied:	63

Number of detected faults:	22
Memory used:	32 Kbytes
CC:	63 Clocks
@ 50 MHz	1.26 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

The speedup between sequential and parallel testing is 22 times faster for both testing methods (deterministic and pseudorandom). This conforms with our theoretical understanding, that the speedup between sequential and parallel testing is equal to the number of fault injections.

4.2. EC13 benchmark circuit

The EC13 circuit is a full-adder circuit. This circuit shows the use of testing functional circuits, as some circuits, as simple as a full-adder, can encompass faults that will be detected. As well, this circuit presents a smaller logic than C17.

The experimental results obtained from hardware sequential testing of a deterministic and a pseudorandom TPG of the EC13 are as follows:

Deterministic sequential testing

	I	O	
Test 1:	101	01	7 faults detected
Test 2:	100	10	5 faults detected
Test 3:	111	11	3 faults detected
Test 4:	001	10	1 faults detected

Test 5: 010 10 0 faults detected

Fault coverage:	100%
Number of test patterns applied:	5
Number of detected faults:	16
Memory used:	32 Kbytes
CC:	80 Clocks
@ 50 MHz	1.6 μs

Localization:
Input: 101 Circuit Output: 01 True Output: 10 Mux-1 Stuck-at-0
Input: 101 Circuit Output: 01 True Output: 11 Mux-2 Stuck-at-1
Input: 101 Circuit Output: 01 True Output: 10 Mux-3 Stuck-at-0
Input: 101 Circuit Output: 01 True Output: 10 Mux-5 Stuck-at-0
Input: 101 Circuit Output: 01 True Output: 00 Mux-6 Stuck-at-0
Input: 101 Circuit Output: 01 True Output: 11 Mux-7 Stuck-at-1
Input: 101 Circuit Output: 01 True Output: 00 Mux-8 Stuck-at-0
Input: 100 Circuit Output: 10 True Output: 01 Mux-3 Stuck-at-1
Input: 100 Circuit Output: 10 True Output: 11 Mux-4 Stuck-at-1
Input: 100 Circuit Output: 10 True Output: 11 Mux-6 Stuck-at-1
Input: 100 Circuit Output: 10 True Output: 00 Mux-7 Stuck-at-0
Input: 100 Circuit Output: 10 True Output: 11 Mux-8 Stuck-at-1
Input: 111 Circuit Output: 11 True Output: 01 Mux-2 Stuck-at-0
Input: 111 Circuit Output: 11 True Output: 10 Mux-4 Stuck-at-0
Input: 111 Circuit Output: 11 True Output: 01 Mux-5 Stuck-at-1
Input: 001 Circuit Output: 10 True Output: 01 Mux-1 Stuck-at-1

Pseudorandom sequential testing

	I	O	
Test 1:	010	10	8 faults detected
Test 2:	000	00	3 faults detected
Test 3:	001	10	1 faults detected
Test 4:	011	01	2 faults detected
Test 5:	111	11	2 faults detected

Fault coverage:	100%
Number of test patterns applied:	63
Number of detected faults:	16
Memory used:	32 Kbytes

CC:	1008 Clocks
@ 50 MHz	20.16 μs

Localization:
Input: 010 Circuit Output: 10 True Output: 01 Mux-1 Stuck-at-1
Input: 010 Circuit Output: 10 True Output: 00 Mux-2 Stuck-at-0
Input: 010 Circuit Output: 10 True Output: 01 Mux-3 Stuck-at-1
Input: 010 Circuit Output: 10 True Output: 11 Mux-4 Stuck-at-1
Input: 010 Circuit Output: 10 True Output: 00 Mux-5 Stuck-at-0
Input: 010 Circuit Output: 10 True Output: 11 Mux-6 Stuck-at-1
Input: 010 Circuit Output: 10 True Output: 00 Mux-7 Stuck-at-0
Input: 010 Circuit Output: 10 True Output: 11 Mux-8 Stuck-at-1
Input: 000 Circuit Output: 00 True Output: 10 Mux-2 Stuck-at-1
Input: 000 Circuit Output: 00 True Output: 10 Mux-5 Stuck-at-1
Input: 000 Circuit Output: 00 True Output: 10 Mux-7 Stuck-at-1
Input: 001 Circuit Output: 10 True Output: 00 Mux-3 Stuck-at-0
Input: 011 Circuit Output: 01 True Output: 00 Mux-6 Stuck-at-0
Input: 011 Circuit Output: 01 True Output: 00 Mux-8 Stuck-at-0
Input: 111 Circuit Output: 11 True Output: 01 Mux-1 Stuck-at-0
Input: 111 Circuit Output: 11 True Output: 10 Mux-4 Stuck-at-0

The experimental results obtained from hardware parallel testing of a deterministic and a pseudorandom TPG of the EC13 are as follows:

Deterministic parallel testing

	I	O	
Test 1:	101	01	7 faults detected
Test 2:	100	02	5 faults detected
Test 3:	111	03	3 faults detected
Test 4:	001	02	1 faults detected
Test 5:	010	10	0 faults detected

Fault coverage:	100%
Number of test patterns applied:	5
Number of detected faults:	16
Memory used:	32 Kbytes
CC:	5 Clocks
@ 50 MHz	0.1 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

Pseudorandom parallel testing

	I	O	
Test 1:	010	10	8 faults detected
Test 2:	000	00	3 faults detected
Test 3:	001	10	1 faults detected
Test 4:	011	01	2 faults detected
Test 5:	111	03	2 faults detected

Fault coverage:	100%
Number of test patterns applied:	63
Number of detected faults:	16
Memory used:	32 Kbytes
CC:	63 Clocks
@ 50 MHz	1.26 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

The speedup between sequential and parallel testing is 16 times faster for both testing methods (deterministic and pseudorandom). Once again, this conforms with our theoretical understanding, that the speedup between sequential and parallel testing is equal to the number of fault injections.

4.3. EC37 benchmark circuit

The EC37 circuit is more complex than the C17 circuit but not as complex as the C432 circuit.

The experimental results obtained from hardware sequential testing of a deterministic and a pseudorandom TPG of the EC37 are as follows:

Deterministic sequential testing

	I	O	
Test 1:	011	1010	7 faults detected
Test 2:	010	1000	9 faults detected
Test 3:	110	1010	2 faults detected
Test 4:	111	1010	1 faults detected

Fault coverage:	100%
Number of test patterns applied:	4
Number of detected faults:	19
Memory used:	32 Kbytes
CC:	160 Clocks
@ 50 MHz	3.2 μs

Localization:
Input: 011 Circuit Output: 1010 True Output: 1000 Mux-3 Stuck-at-0
Input: 011 Circuit Output: 1010 True Output: 0010 Mux-4 Stuck-at-1
Input: 011 Circuit Output: 1010 True Output: 0010 Mux-13 Stuck-at-0
Input: 011 Circuit Output: 1010 True Output: 0010 Mux-17 Stuck-at-0
Input: 011 Circuit Output: 1010 True Output: 1110 Mux-18 Stuck-at-1
Input: 011 Circuit Output: 1010 True Output: 1000 Mux-19 Stuck-at-0
Input: 011 Circuit Output: 1010 True Output: 1011 Mux-20 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-1 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-3 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-5 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-7 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-10 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-11 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-14 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-16 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-19 Stuck-at-1

Input: 110 Circuit Output: 1010 True Output: 1000 Mux-1 Stuck-at-0
Input: 110 Circuit Output: 1010 True Output: 1000 Mux-7 Stuck-at-1
Input: 111 Circuit Output: 1010 True Output: 0010 Mux-9 Stuck-at-1

Pseudorandom sequential testing

	I	O	
Test 1:	010	1000	12 faults detected
Test 2:	101	1010	2 faults detected
Test 3:	100	1010	2 faults detected
Test 4:	011	1010	2 faults detected
Test 5:	111	1010	1 faults detected

Fault coverage:	100%
Number of test patterns applied:	60
Number of detected faults:	19
Memory used:	32 Kbytes
CC:	2400 Clocks
@ 50 MHz	48 μs

Localization:
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-1 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-3 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-5 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-7 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-10 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-11 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1011 Mux-14 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-16 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 0000 Mux-17 Stuck-at-0
Input: 010 Circuit Output: 1000 True Output: 1100 Mux-18 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1010 Mux-19 Stuck-at-1
Input: 010 Circuit Output: 1000 True Output: 1001 Mux-20 Stuck-at-1
Input: 101 Circuit Output: 1010 True Output: 0010 Mux-13 Stuck-at-0
Input: 101 Circuit Output: 1010 True Output: 1000 Mux-19 Stuck-at-0
Input: 100 Circuit Output: 1010 True Output: 1000 Mux-1 Stuck-at-0
Input: 100 Circuit Output: 1010 True Output: 1000 Mux-7 Stuck-at-1
Input: 011 Circuit Output: 1010 True Output: 1000 Mux-3 Stuck-at-0
Input: 011 Circuit Output: 1010 True Output: 0010 Mux-4 Stuck-at-1
Input: 111 Circuit Output: 1010 True Output: 0010 Mux-9 Stuck-at-1

The experimental results obtained from hardware parallel testing of a deterministic and a pseudorandom TPG of the EC37 are as follows:

Deterministic parallel testing

	I	O	
Test 1:	011	1010	7 faults detected
Test 2:	010	1000	9 faults detected
Test 3:	110	1010	2 faults detected
Test 4:	111	1010	1 faults detected

Fault coverage:	100%
Number of test patterns applied:	4
Number of detected faults:	19
Memory used:	32 Kbytes
CC:	4 Clocks
@ 50 MHz	0.08 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

Pseudorandom parallel testing

	I	O	
Test 1:	010	1000	12 faults detected
Test 2:	101	1010	2 faults detected
Test 3:	100	1010	2 faults detected
Test 4:	111	1010	1 faults detected
Test 5:	011	1010	2 faults detected

Fault coverage:	100%
Number of test patterns applied:	60
Number of detected faults:	19
Memory used:	32 Kbytes
CC:	60 Clocks
@ 50 MHz	1.2 μs

Localization:
The localization of the fault causing wires is the same as sequential testing.

The speedup between sequential and parallel testing is 40 times faster for both testing methods (deterministic and pseudorandom). Once again, this conforms with our theoretical understanding, that the speedup between sequential and parallel testing is equal to the number of fault injections.

4.4. C432 benchmark circuit

The experimental results obtained from software testing of a deterministic and a pseudorandom TPG of the C432 are as follows:

Deterministic testing (according to ATALANTA)

	I n p u t	Output	
Test 1:	10101000011011111001001011110001 1010	1011010	64 faults detected
Test 2:	10100001001100111111111010000000 1001	0001100	90 faults detected
Test 3:	10001110100100101110000010001000 0111	1011100	20 faults detected
Test 4:	11111001000101001101111100101111 0111	1101100	17 faults detected
Test 5:	10011000100100111101110001001100 1000	1101010	13 faults detected
Test 6:	11001001011011110110001111000100 1110	1101001	33 faults detected
Test 7:	11101111111101111011011111010111 1110	1001111	25 faults detected
Test 8:	00011011001111010110001001001011 1011	1101010	7 faults detected
Test 9:	11010111011000000110010110100001 1100	1101011	24 faults detected
Test10:	01100010111100011111000111111111 0110	1010110	19 faults detected
Test 11:	00110100101111100010011000010010 0101	1101000	22 faults detected
Test 12:	11010001010000101011010100010110 1011	0110110	13 faults detected
Test 13:	00111000000000001011100010010110 1001	0001001	11 faults detected
Test 14:	00001110000101110000111110101000 1000	1011111	9 faults detected
Test 15:	10001011011010111011001010011110 0111	1001000	7 faults detected
Test 16:	10011010011101111000001010111010 0100	1011110	11 faults detected

Test 17: 00010001100101101101001000000011111111 1111100 1 faults detected
Test 18: 11011100100100111111000111100001001 1 0100111 9 faults detected
Test 19: 11011111000010010010101010111000000 1011001 6 faults detected
Test 20: 00001110010011111110001010101000001 1 1101110 7 faults detected
Test 21: 10110111010111010111100000110100100 0 0111101 17 faults detected
Test 22: 11100010000100001111111010010010110 1001100 4 faults detected
Test 23: 01101110011111101101010011010101001 1 1111100 1 faults detected
Test 24: 01111110110011100110111011001001110 1101110 2 faults detected
Test 25: 10111001000100110011010000011110100 0 0111011 6 faults detected
Test 26: 11110010101110011011000101011001100 0 0101010 5 faults detected
Test 27: 10001011111011000110001001010100110 1 1111101 3 faults detected
Test 28: 11011010101010100001010111110101001 0 0111001 5 faults detected
Test 29: 00111011010110010101000110000110111 0 1011000 9 faults detected
Test 30: 00000011001010111100100111111101110 0 0111000 5 faults detected
Test 31: 00010111011100111111101111001110101 1 1001001 3 faults detected
Test 32: 10011101100010010111100010011000101 0 0101111 3 faults detected
Test 33: 11110111011101010100111000000011011 0 1011011 4 faults detected
Test 34: 10100000101111010100000111000111100 0 1001001 3 faults detected
Test 35: 00100101001011101001001100001000000 1 1111111 1 faults detected
Test 36: 10111001100101000011010101111001000 1 0111101 1 faults detected
Test 37: 01110001000110010011000110110010000 0 1010000 7 faults detected
Test 38: 00011001010011001001011000000011110 1 1101101 6 faults detected
Test 39: 01100101101110111110100001011100110 0 1111000 1 faults detected
Test 40: 00000100110001000110000100110011000 1 1111110 1 faults detected
Test 41: 00010011001110111100100100010110100 1 0101100 1 faults detected
Test 42: 11110100000001100111100000110011111 0 1011101 3 faults detected
Test 43: 00100010011100110001110101111110100 1011110 1 faults detected
Test 44: 10010011110100111111011111011000000 1 0111110 3 faults detected
Test 45: 11010000101110110000000111111111111 1 0110000 4 faults detected
Test 46: 11001111110110000011100001000001100 1 1111010 1 faults detected
Test 47: 00111010011001011010011000000111001 0 1101101 2 faults detected
Test 48: 00000000000000000000000000000000000 0 0000000 10 faults detected

Fault coverage:	99.237%
Number of test patterns applied:	48
Number of detected faults:	524
Memory used:	2499 Kbytes
CPU time:	0.200 seconds

Pseudorandom testing (according to FSIM)

	Input	Output	
Test 1:	00000000000000000000000000000000000000	0000000	122 faults detected
Test 2:	01100000101111001000001001001100	1111000	33 faults detected
Test 3:	11101010000000011111111010011101110	0101001	38 faults detected
Test 4:	01100001111011110111001101011000101	1101000	20 faults detected
Test 5:	10101111011000001010111011001001110	1101001	14 faults detected
Test 6:	10010010001110100111000000011001101	1011100	25 faults detected
Test 7:	11111110101010011011010011100110011	1011010	18 faults detected
Test 8:	10011111010101101010001101010010011	1001000	25 faults detected
Test 9:	10100001010100111011001010011101111	0111110	17 faults detected
Test 10:	00010110011100110111111101111100111	1011110	11 faults detected
...			
... (The rest of the test results are in APPENDIX A)			
...			
Test 598:	00001110101010111111110111010111011	1001111	0 faults detected
Test 599:	01100001101100010001001000011010100	1010000	0 faults detected
Test 600:	00000110110100001111100010111100111	0	1111110 faults detected
Test 601:	10010000101110011101111100001111101	0	1001001 faults detected
Test 602:	10111100011110011010000001000000100	0	1111010 faults detected
Test 603:	11011000000110101001000001101110000	1	1011001 faults detected
Test 604:	10110111010111000000100101110011111	1	0111101 faults detected
Test 605:	01110111100101001000001011100101000	1	1111001 faults detected
Test 606:	10111110001101100101011111010100101	1	1111100 faults detected
Test 607:	10111010010010100110000011101010011	0	1101110 faults detected
Test 608:	00101000100111100111111000100010000	0	1001100 faults detected

Fault coverage:	99.237%
Number of test patterns applied:	608
Number of detected faults:	520 plus 4 undetected faults
Memory used:	463 Kbytes
CPU time:	0.083 seconds

The above results were accomplished using and software simulators ATLANTA and FSIM, both of which don't provide the fault occurrence (i.e. at which line did the faults occur). However, in our design we will extract the exact line that caused the faults. Moreover, our design will reveal if the occurring fault was a stuck-at-0 or a stuck-at-1 fault.

The experimental results obtained from hardware sequential testing of a deterministic and a pseudorandom TPG of the C432 are as follows:

Deterministic sequential testing

	I (in hex)	O (in hex)	
Test 1:	A133FE8090	0C	115 faults detected
Test 2:	8E92E08870	5C	23 faults detected
Test 3:	F914DF2F70	6C	13 faults detected
Test 4:	9893DC4C80	6A	19 faults detected
Test 5:	C96F63C4E0	69	28 faults detected
Test 6:	EFF7BBEBE0	4F	14 faults detected
Test 7:	1B3D624BB0	6A	8 faults detected
Test 8:	D76065A1C0	6B	19 faults detected
Test 9:	62F1F1FF60	56	18 faults detected
Test 10:	34BE261250	68	18 faults detected
Test 11:	D142B516B0	36	6 faults detected
Test 12:	3800B89690	09	2 faults detected
Test 13:	0E170FA880	5F	8 faults detected
Test 14:	8B6BB29E70	48	3 faults detected
Test 15:	9A7782BD40	5E	9 faults detected
Test 16:	DC93F1E130	27	5 faults detected
Test 17:	DF092AAE00	59	6 faults detected
Test 18:	0E4FE2A830	6E	5 faults detected
Test 19:	B75D783480	3D	19 faults detected
Test 20:	7ECE6F64E0	6E	1 faults detected
Test 21:	B913341E80	3B	6 faults detected
Test 22:	F2B9B15980	2A	1 faults detected
Test 23:	8BEC6254D0	7D	1 faults detected
Test 24:	DAAA15F520	39	3 faults detected
Test 25:	3B595186E0	58	8 faults detected
Test 26:	032BC9FDC0	38	1 faults detected
Test 27:	1773FBCEB0	49	1 faults detected
Test 28:	9D897898A0	2F	2 faults detected
Test 29:	711931B200	50	9 faults detected
Test 30:	194C9603D0	6D	3 faults detected
Test 31:	65BBE85CC0	78	1 faults detected
Test 32:	D0BB01FFF0	30	2 faults detected
Test 33:	CFD8384190	7A	6 faults detected
Test 34:	3A65A60720	6D	2 faults detected
Test 35:	0000000000	00	3 faults detected

Fault coverage:	98.98%
Number of test patterns applied:	48
Number of detected faults:	388
Memory used:	32 Kbytes
CC:	18816 Clocks
@ 50 MHz	376.32 μs

Localization:
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-2 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 7F Mux-6 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 1E Mux-10 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-14 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 4C Mux-16 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-18 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 2C Mux-19 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 4B Mux-20 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 1C Mux-21 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 2B Mux-23 Stuck-at-0
...
... (The rest of the test results are in APPENDIX B)
...
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-77 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-119 Stuck-at-1
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-129 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-144 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-159 Stuck-at-1
Input: 3A65A60720 Circuit Output: 6D True Output: 70 Mux-137 Stuck-at-1
Input: 3A65A60720 Circuit Output: 6D True Output: 6E Mux-143 Stuck-at-1
Input: 0000000000 Circuit Output: 00 True Output: 30 Mux-46 Stuck-at-0
Input: 0000000000 Circuit Output: 00 True Output: 08 Mux-182 Stuck-at-0
Input: 0000000000 Circuit Output: 00 True Output: 08 Mux-184 Stuck-at-0

Pseudorandom sequential testing

	I (in hex)	O (in hex)	
Test 1:	183A183AA0	00	105 faults detected
Test 2:	18FE18FEE0	58	37 faults detected
Test 3:	016F016FF0	5D	23 faults detected
Test 4:	E326E32660	5D	19 faults detected

Test 5:	C712C71220	7C	17 faults detected
Test 6:	3639363990	1F	15 faults detected
Test 7:	7FDB7FDBB0	40	18 faults detected
Test 8:	E3B1E3B110	14	8 faults detected
Test 9:	264B264BB0	6E	11 faults detected
Test 10:	6F956F9550	47	11 faults detected
Test 11:	F8BCF8BCC0	78	3 faults detected
Test 12:	63B763B770	50	8 faults detected
Test 13:	80F280F220	5E	1 faults detected
Test 14:	B802B80220	00	2 faults detected
Test 15:	46CF46CFF0	70	4 faults detected
Test 16:	B806B80660	59	2 faults detected
Test 17:	A3FDA3FDD0	2D	4 faults detected
Test 18:	DFC6DFC660	6C	4 faults detected
Test 19:	8DB68DB660	6F	1 faults detected
Test 20:	1C4A1C4AA0	6E	2 faults detected
Test 21:	108D108DD0	7D	2 faults detected
Test 22:	F774F77440	78	1 faults detected
Test 23:	4ED74ED770	7E	1 faults detected
Test 24:	3874387440	78	1 faults detected
Test 25:	E5DCE5DCC0	78	2 faults detected
Test 26:	C2F5C2F550	5E	2 faults detected
Test 27:	8005800550	79	12 faults detected
Test 28:	4EF14EF110	60	3 faults detected
Test 29:	88EC88ECC0	78	1 faults detected
Test 30:	4CAF4CAFF0	67	1 faults detected
Test 31:	79B679B660	58	1 faults detected
Test 32:	E3DFE3DFF0	2E	3 faults detected

Fault coverage:	82.91%
Number of test patterns applied:	779
Number of detected faults:	325
Memory used:	32 Kbytes
CC:	305368 Clocks
@ 50 MHz	6107.36 μs

Localization:
Input: 183A183AA0 Circuit Output: 00 True Output: 60 Mux-2 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 3F Mux-6 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 4E Mux-10 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 1D Mux-14 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 6C Mux-18 Stuck-at-1

Input: 183A183AA0 Circuit Output: 00 True Output: 3B Mux-22 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 4A Mux-26 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 09 Mux-30 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 58 Mux-34 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-38 Stuck-at-0
...
... (The rest of the test results are in APPENDIX C)
...
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-160 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 70 Mux-137 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 70 Mux-144 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 63 Mux-149 Stuck-at-1
Input: 88EC88ECC0 Circuit Output: 78 True Output: 79 Mux-172 Stuck-at-1
Input: 4CAF4CAFF0 Circuit Output: 67 True Output: 70 Mux-138 Stuck-at-1
Input: 79B679B660 Circuit Output: 58 True Output: 5C Mux-169 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 30 Mux-135 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 26 Mux-139 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 30 Mux-140 Stuck-at-1

To compare our hardware expected results with software results, we have executed the c17 using the ATALANTA simulator, with 48 test inputs (deterministic testing). The simulator needed a total time of 200 ms to finish execution. Our hardware needs 18816 CC for deterministic testing, which translates to 0.376 ms, with a 50 MHz clock. A speedup of 53 x 10. And our hardware needs 305368 CC for random testing, which translates to 6.1ms, with a 50 MHz clock. A speedup of 33 x 10.

Nevertheless, these speedups can even be improved with a parallel hardware implementation at the cost of memory. The improvement would generate a speedup of 392, which is the number of fault injection in the C432 circuit. Thence, deterministic parallel testing would require 18816 / 392 = 48 CC, which translates to 0.96 μs at 50 MHz. And, random parallel testing would require 305368 / 392 = 779 CC, which translates to 15.58 μs at 50 MHz.

Unfortunately, the appropriate tools weren't accessible to prove this design for a large scale circuit, as the C432. Nonetheless, parallel testing increases execution speed by a factor of the number of fault injection in the given circuit, and this has been illustrated in ec13 and ec37 circuits.

Chapter 5

5. Conclusion and work ahead

In this thesis, a novel co-processor architecture was introduced. It is targeted for adaptive embedded system applications, and is capable of handling variant functions (arithmetic, DSP, MAC, etc...). A task is available for the RPU to use once it is implemented in HDL and its respective bitstream is generated. The entire system is being implemented on a System-on-Chip (SoC), with a soft-core microprocessor handling all real-time deadlines and a reconfigurable co-processor allowing for the parallel execution of multiple hardware functional units, with the help of a just-in-time compiler that manages both the application and reconfiguration flows.

In our work we have utilized the abovementioned architecture and presented a novel approach to digital circuit testing using BIST techniques on an RPU. The approach has been compared to traditional software testing (software simulators) and embedded software testing. The dominance of the new approach has been demonstrated on four circuits, with great speedups.

As it was demonstrated, all of the tests achieved 100% fault coverage with the exception of C432 circuit, reaching 82.91% fault coverage for pseudorandom sequential testing. However, the reason for not reaching 100%, the test was interrupted after 780 test patterns have been applied. The interruption was applied because we were comparing our results with other literatures' results which have tested approximately 780 test patterns. Given the two approaches (software and sequential hardware) were utilizing the same seed for random

number generation the fault coverage would be similar. Moreover, hardware testing has achieved superior fault coverage than its counterpart software testing. Additionally, there were no variances in fault coverage, both achieved identical results, between sequential hardware testing and parallel hardware testing, proving once again, that parallel hardware testing is simply a trade-off of execution speed to memory usage.

Furthermore, the speedups between software and sequential hardware testing, as noted in the results chapter, were as large as 1×10^4. However, what is worth noting is that since these CUTs are combinational circuits, hardware testing will not decrease in performance while complexity of the CUTs increase, on the contrary, software suffers in performance as the complexity of the CUTs grows, given hardware testing an upper hand in digital circuit testing and a brighter future.

Lastly, we observe, when comparing parallel versus sequential hardware testing, parallel testing will always outperform sequential testing by a factor of the number of fault injections (hence the number of wires in a circuit), therefore the larger the circuit the larger the factor of speedup between parallel and sequential hardware testing. Consequence, parallel testing requires more logic utilization. For example when comparing the EC37 parallel deterministic testing versus sequential deterministic testing, while keeping in mind that the EC37 circuit has 40 fault injections. We notice that sequential deterministic testing only utilizes 714 (6%) out of 10752 slices and 729 (3%) out of 21504 slice flip flops and 1194 (5%) out of 21504 4-input LUTs. While, its counterpart, parallel deterministic testing, utilizes 759 (7%) out of 10752 slices and 735 (3%) out of 21504 slice flip flops and 1280 (5%) out of 21504 4-input LUTs. From those numbers we confirm the rise in logic utilization in all 3 aspects (slices, slice flip flops, and 4-

input LUTs), however at this small rise in cost performance is increase by a factor of 40.

Beneficial improvements to the architecture include adjustments of how the retrieval of the test parameters and results are achieved. One option is to utilize the already employed Fast Simplex Links (FSLs) between the hardware block and MicroBlaze. The FSL is a unidirectional buffer that can operate using 8-, 16-, or 32-bit data transfers, with a buffer length limited by the number of Block RAMs (BRAMs) available on the particular reconfigurable device (in this case, the Virtex-II contains 56 BRAMs, totally 112 kB of total addressable on-chip RAM). This added module (FSL) relieves congestion on the bus while other tasks are simultaneously being executed and utilizing the bus.

Another valuable improvement is to reduce the number of outputs outputted by any CUT by utilizing the previously mentioned space compaction compressors. These compressors are zero-aliasing, as all the faults that can be detected at the outputs of the CUT, can also be detected at the outputs of the CUT and the compressor combined. The compressor reduces the number of system outputs, which would provide us with a smaller number of outputs to first store, and then compare.

An additional favourable improvement is to embark on testing the other two benchmark circuit types, ISCAS'89 and ISCAS'99. ISCAS'89 benchmark circuits poses a challenge as the benchmark circuits it provides are not combination, they are sequential, hence challenging to detect when the circuit has completed testing of the circuit, and speedups might not be as notable as the combinational ISCAS'85 circuits. Nevertheless, once improvements for testing ISCAS'89 circuits have been completed, the next and final milestone for circuit testing is to go on board with testing ISCAS'99. ISCAS'99 benchmark circuits are complete

cores with both sequential and combinational circuits coupled together. Digital core testing would the testing field's ultimate goal, because the testing architecture can employ their system to test any core of any shape and size. This sort of system would be utilized in factory lines as the final stop to examine if the ICs enclose any stuck-at faults.

The work ahead of this thesis is vast for both the application and architecture. As this thesis concentrated on the stuck-at fault testing as the application, other fields can easily take advantage of this architecture by employ their own specific hardware blocks that essentially act as hardware accelerators, fields such as DSP, Digital Image Processing (DIP), etc....

BIBLIOGRAPHY

[1] Parag K. Lala, Digital Circuit Testing and Testability, San Diego, CA: Academic Press, 1997.

[2] S. R. Das, E. M. Petriu, T. Barakat, M. H. Assaf, and A. R. Nayak, "Space compaction under generalized mergeability," IEEE Transactions on Instrumentation and Measurement, pp. 1283 – 1293, October 1998.

[3] E. J. McCluskey, "Built-in self-test techniques," IEEE Design and Test of Computes, pp. 21 – 28, April 1983.

[4] K. K. Saluja and M. Karpovsky, "Testing computer hardware through compression in space and time," Proceedings of International Test Conference, 1983, pp. 83 – 88.

[5] D. K. Pradhan and S. K. Gupta, "A new framework for designing and analyzing BIST techniques and zero aliasing compression," IEEE Transactions on Computers, pp. 743 – 763, June 1991.

[6] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "Compactest: A method to generate compact test sets for combinational circuits," Proceedings of International Test Conference, 1991, pp. 194 – 203.

[7] S. M. Reddy, K. K. Saluja, and M. G. Karpovsky, "Data compression technique for test responses," IEEE Transactions on Computers, pp. 1151 – 1156, September 1988.

[8] Mansour H. Assaf, Rami S. Abielmona, Payam Abolghasem, Sunil R. Das, Emil M. Petriu, Voicu Groza, and Mehmet Sahinoglu, "Test Implementation of Embedded Cores-Based Digital Devices in JBits Java Simulation Environment," Lecture Notes in Computer Science, Vol. 3356, January 2004, pp. 315 – 325.

[9] Wikipedia. (2006, May 13). Moore's law. [Online]. Available: http://en.wikipedia.org/wiki/Moore's_law.

[10] P. H. Bardell, W. H. McAnney, and J. Savir, "Built-in Test for VLSI: Pseudorandom Techniques," New York, NY: Wiley, 1987.

[11] K. Chakrabarty, "Test Response Compaction for Built-In Self-Testing," doctoral dissertation, Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, 1995.

[12] R. Leveugle, "Fault injection in VHDL descriptions and emulation," In Proceedings of the 15th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 25 – 27 October 2000, pp. 414 – 419.

[13] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer, "Fault Injection Techniques and Tools," IEEE Computer, vol. 30, (4), pp. 75 – 82, 1997.

[14] A. Parreira, J. Teixeira, and M. Santos, "A Novel Approach to FPGA-based Hardware Fault Modeling and Simulation," In Proceedings of the Design and Diagnostics of Electronic Circuits and System Workshop, April 2003, pp. 17 – 24.

[15] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda, and M. Violante, "An FPGA-Based Approach for Speeding-up Fault Injection Campaigns on Safety-Critical Circuits," In IEEE Journal of Electronic Testing Theory and Applications, vol. 18, (3) , pp. 261 – 271, June 2002.

[16] Xilinx, (September 2004). Two Flows for Partial Reconfiguration: Module Based or Difference Based. [Online]. XAPP 290. Available: http://www.xilinx.com/bvdocs/appnotes/xapp290.pdf

[17] Technická Univerzita V Liberci. (2006, February 27). Benchmark Circuits. [Online]. Available: http://www.fm.vslib.cz/~kes/asic/iscas/

[18] H. K. Lee and D. S. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation," Proc. 1991 Int. Conference, 1991, pp. 946 – 955.

[19] H. K. Lee and D. S. Ha, On the Generation of Test Patterns for Combinational Circuits. Technical Report No. 12 – 93, Dept. of Electrical Eng. Virginia Polytechnic Institute and State University.

[20] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," Proc. Design Automation Conference, June 1992, pp. 336 – 340.

[21] Mansour Hanna Assaf, "Digital core output test data compression architecture based on switching theory concepts: model implementation and analysis," doctoral thesis, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada, 2003.

[22] Xilinx. (2006, May 14). Xilinx Multimedia Board. [Online]. Available: http://www.xilinx.com/products/boards/multimedia/

[23] A. Parreira, J. P. Teixeira, A. Pantelimon, M. B. Santos, and J.T. de Sousa, "Fault Simulation using Partially Reconfigurable Hardware," In the 13th International Conference on Field Programmable Logic and Applications (FPL'03), Lisbon, Portugal, 1 – 3 September 2003, pp. 839 - 848.

[24] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection in hardware prototypes," In IEEE International Symposium

Defect and Fault Tolerance in VLSI Systems, Yamanashi, 25 – 27 October 2000, pp. 405 – 413.

[25] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection applications," In Proceedings of 18th IEEE Instrumentations and Measurement Technology Conference 2001 (IMTC 2001), volume 3. Budapest, Hungary, May 2001, pp. 1773 – 1777.

[26] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection in hardware prototypes," In Proceedings of 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'02), 6 – 8 November 2002, pp. 245 – 253.

[27] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection applications," In IEEE Transaction on Instrumentations and Measurement, vol. 52, (5), pp. 1468 – 1473, October 2003.

[28] V. Groza, R. Abielmona, M. El-Kadri, N. Sakr, and M. Elbadri, "A Reconfigurable Co-Processor for Adaptive Embedded Systems," WISES 2004 - Workshop on Intelligent Solutions in Embedded Systems, Graz, Austria, June 25, 2004, pp. 105 – 115.

[29] V. Groza, N. Sakr, and M. Elbadri, "Run-Time Reconfigurable System-on-Chip," IMTC 2005 - IEEE Instrumentation and Measurement Technology Conference, Ottawa, Ontario, Canada, May 17-19, 2005, pp. 322 – 326.

[30] M. Elbadri, V. Groza, R. Abielmona, M. Assaf, "A Reconfigurable Processing Unit for Digital Circuit Testing using Built-In Self-Test Techniques," IMTC 2006 – IEEE Instrumentation and Measurement Technology Conference, Sorrento, Italy, April 24 – 27, 2006.

[31] V. Groza, R. Abielmona, M. Elbadri, M. El-Kadri, P. Fu, and N. Sakr, "A Reconfigurable Co-Processor for Adaptive Embedded Systems," TEXPO 2004 – CMC symposium, Ottawa, Ontario, Canada, September 20, 2004.

[32] M. El-Kadri, M. Elbadri, R. Abielmona, V. Groza, "Run-Time Reconfigurable Digital Core Testing for Biomedical Instrumentation," TEXPO 2005 – CMC symposium, Ottawa, Ontario, Canada, October 13, 2005.

[33] Michael L. Bushnell and Vishwani D. Agrawal, Essentials of Electronic Testing: for Digital, Memory & Mixed-Signal VLSI Circuits, Norwell, MA: Kluwer Academic Publishers, 2000.

[34] Rami Abielmona, "Circuit synthesis evolution using a hardware-based genetic algorithm," master's thesis, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada, 2002.

[35] G. Aggarwal, N. Thaper, K. Aggarwal, M. Balakrishnan, and S. Kumar, "A Novel Reconfigurable Co-Processor Architecture," In Proceedings of Tenth International Conference on VLSI Design, Hyderabad, 4 – 7 January 1997, pp. 370 – 375.

[36] G. Haug and W. Rosenstiel, "Reconfigurable Hardware as Shared Resource in Multipurpose Computers," In Field-Programmable Logic: From FPGAs to Computing Paradigm, editors, Reiner W. Hartenstein and Andres Keevallik, Springer-Verlag, Berlin, August/September 1998, pp. 149 – 158.

[37] Xilinx. (2003, October 14). Virtex-II Platform FPGAs: Complete Data Sheet. [Online]. DS031. Available: http://www.xilinx.com/partinfo/ds031.pdf

[38] D. Wo and K. Forward, "Compiling to the Gate Level for a Reconfigurable Co-Processor," In Proceeding of FPGAs for Custom Computing Machines, Napa Valley, CA, 10 – 13 April 1994, pp. 147 – 154.

APPENDIX A

The following is the output results for pseudorandom testing (according to FSIM) of the C432 benchmark circuit (the test patterns that yielded zero fault detection were omitted):

	Input	Output	
test	1: 10101000011011111001001011100011010	1011010	64 faults detected
test	2: 10100001001100111111111010000000001001	0001100	90 faults detected
test	3: 10001110100100101110000010001000111	1011100	20 faults detected
test	4: 11111001000101001101111001011110111	1101100	17 faults detected
test	5: 10011000100100111101110001001100100 0	1101010	13 faults detected
test	6: 11001001011011101100011110001001110	1101001	33 faults detected
test	7: 11101111111101111011011111010111110	1001111	25 faults detected
test	8: 00011011001111010110001001001011101 1	1101010	7 faults detected
test	9: 11010111011000000110010110100001110 0	1101011	24 faults detected
test	10: 01100010111100011111000111111111010	1010110	19 faults detected
test	11: 10110010101110101101000101010010 11	1111100	6 faults detected
test	12: 00100010101110111111011111111010111 1	1001011	12 faults detected
test	13: 10110101000111011000111001111100000 1	1011011	15 faults detected
test	14: 01101100100100111101000100000011000 0	1101111	21 faults detected
test	15: 11011101011110011100100111100110110 1	1101000	2 faults detected
test	16: 11010100111000110111100100101111110 0	1011110	4 faults detected
test	17: 11001001001101111101000100111110011 0	1011000	2 faults detected
test	18: 11010010101101001011000111000001101 0	0110010	12 faults detected
test	19: 10011001101010011111010110000101111 0	1101001	2 faults detected
test	22: 11010100111111000111000100010111011 0	1011100	2 faults detected
test	24: 01000110111001100100111001101010000 0	1110000	11 faults detected
test	26: 11100000100001111010100100100110101 1	1001101	11 faults detected
test	28: 11010011010011010000001010101110000 0	1111101	13 faults detected
test	30: 00110011010011111101010110001011011 1	1001101	2 faults detected
test	31: 00110100111110101100001001000111100 0	1111100	1 faults detected
test	32: 11010110001011101100111001101000101 0	1101100	2 faults detected
test	34: 01011001101110001100101101101001110 1	1100100	4 faults detected
test	35: 00101010111011001110010010100010111 1	1101101	2 faults detected
test	38: 00001100011010101100101010101100011	1111111	2 faults detected
test	39: 01010101011011100001010010110110011	1110000	1 faults detected
test	40: 11111111110010100110010100011011110 0	1111011	5 faults detected
test	41: 00100100000110100011111110011010100 1	1111111	3 faults detected
test	50: 00001001100110111100011011010011101 0	1111010	10 faults detected
test	51: 11110101111100000011000011000000001 0	1111010	1 faults detected
test	52: 11000001100001110111100100111111101 1	1011101	1 faults detected

test 53: 1111010100101100001100100100000100011 1111101 1 faults detected
test 54: 1100101100111000111110011010100101101 1001100 1 faults detected
test 57: 0111011101101110010001000000001100100 1111100 1 faults detected
test 58: 0010000100010111101000010011000100010 0001101 1 faults detected
test 61: 1011101001011001001110000000011111001 1101110 3 faults detected
test 62: 1000011000100111010100101111000011001 1111000 1 faults detected
test 63: 1100000001001100110110100011011010101 1101110 1 faults detected
test 64: 1101100111101000110111100101111100001 1101100 1 faults detected
test 68: 1011110010110000000100111011100100000 0101111 2 faults detected
test 69: 1011110101111110011010001111101010111 1001100 1 faults detected
test 70: 0100101010110100101110101100000000000 1111010 1 faults detected
test 71: 0111101111110000011110101110010001101 1111001 5 faults detected
test 72: 1100001000000011000101101011000110100 0110000 1 faults detected
test 84: 1010000111000000001010100101100000101 1101010 1 faults detected
test 92: 1111010001011000010110101110010110101 1101110 3 faults detected
test 94: 0100010111001011011111111001110110010 1110000 1 faults detected
test 99: 1001100111111111011001111110011110010 1101100 2 faults detected
test 101: 1000100011000010000001010111100011001 1111110 1 faults detected
test 104: 0011100101010101110001000001000100001 1111011 1 faults detected
test 106: 0110010011110100001101010011100010011 1101111 1 faults detected
test 114: 1000011010101111110100000000011101101 1011001 2 faults detected
test 115: 0111010111110011000010000011000001101 1010000 1 faults detected
test 117: 0000111111100001011011101000001111111 1001111 1 faults detected
test 126: 1010100000010011111110111111110111100 0111000 2 faults detected
test 136: 0001000001000001001000010001101101001 1111110 1 faults detected
test 143: 1111111000110100111111110101111101001 1001100 2 faults detected
test 151: 1000111100110010111110010000111111101 1011111 1 faults detected
test 173: 0001000001101111101111000111011101101 1011010 1 faults detected
test 178: 1111101010100100000011110010011011 0101010 1 faults detected
test 190: 0110011010011001011111011010001101101 1010000 1 faults detected
test 212: 0001000110111111010110011111111000011 0101100 2 faults detected
test 219: 0101001000111001000101011010001100001 1110000 1 faults detected
test 226: 0100010111010000101001011110000000101 1110000 1 faults detected
test 246: 1001011000111100000101110001111100101 0111101 1 faults detected
test 273: 1000000100111000100111110110101000110 0011011 2 faults detected

APPENDIX B

The following is the localization output results for deterministic sequential and parallel testing of the C432 benchmark circuit (the test patterns that yielded zero fault detection were omitted):

Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-2 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 7F Mux-6 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 1E Mux-10 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-14 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 4C Mux-16 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-18 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 2C Mux-19 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 4B Mux-20 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 1C Mux-21 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 2B Mux-23 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 1B Mux-25 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 7A Mux-26 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 69 Mux-30 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 68 Mux-34 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-38 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 30 Mux-40 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-42 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-45 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-47 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 30 Mux-50 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-52 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-54 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-55 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-56 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-57 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-58 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-59 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-60 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-61 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-62 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-63 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-64 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-65 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-66 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-67 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-68 Stuck-at-1

Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-69 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-70 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-71 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-72 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-73 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-74 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-75 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-76 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-77 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-78 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-79 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-80 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-81 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 40 Mux-82 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 00 Mux-84 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 4C Mux-85 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-99 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-104 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-105 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-106 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-107 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-108 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-109 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-110 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-111 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-112 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-113 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-114 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-115 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-116 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-117 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-118 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-119 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-120 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-121 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 20 Mux-122 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-123 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-124 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-125 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-126 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-127 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-128 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-129 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-130 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-131 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 00 Mux-133 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 2C Mux-134 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-147 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-153 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-154 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-155 Stuck-at-0

Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-156 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-157 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-158 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-159 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-160 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-161 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 10 Mux-162 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 00 Mux-163 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 1C Mux-164 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-169 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 04 Mux-174 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0F Mux-175 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0E Mux-176 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-177 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0B Mux-178 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-181 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 04 Mux-183 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 04 Mux-184 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-185 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-188 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 04 Mux-189 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-190 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0F Mux-191 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0E Mux-192 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-193 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 08 Mux-194 Stuck-at-0
Input: A133FE8090 Circuit Output: 0C True Output: 0E Mux-195 Stuck-at-1
Input: A133FE8090 Circuit Output: 0C True Output: 0D Mux-196 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 6F Mux-7 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 5F Mux-9 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-16 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-21 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 6B Mux-22 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 68 Mux-35 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-45 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-46 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-61 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-73 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 0F Mux-82 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-83 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 1C Mux-85 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-90 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-117 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-127 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-132 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-140 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-157 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 4F Mux-162 Stuck-at-1
Input: 8E92E08870 Circuit Output: 5C True Output: 5F Mux-163 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 4C Mux-164 Stuck-at-0
Input: 8E92E08870 Circuit Output: 5C True Output: 5F Mux-166 Stuck-at-1

Input: F914DF2F70 Circuit Output: 6C True Output: 6D Mux-12 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 59 Mux-19 Stuck-at-1
Input: F914DF2F70 Circuit Output: 6C True Output: 79 Mux-31 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 59 Mux-72 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 6D Mux-84 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 6D Mux-98 Stuck-at-1
Input: F914DF2F70 Circuit Output: 6C True Output: 59 Mux-109 Stuck-at-1
Input: F914DF2F70 Circuit Output: 6C True Output: 59 Mux-122 Stuck-at-1
Input: F914DF2F70 Circuit Output: 6C True Output: 70 Mux-132 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 6D Mux-133 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 4C Mux-134 Stuck-at-0
Input: F914DF2F70 Circuit Output: 6C True Output: 70 Mux-146 Stuck-at-1
Input: F914DF2F70 Circuit Output: 6C True Output: 6D Mux-151 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-24 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-26 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-27 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 7A Mux-29 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 79 Mux-33 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 70 Mux-91 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 6C Mux-99 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 6B Mux-100 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-101 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-150 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-171 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 6C Mux-178 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 6B Mux-179 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 69 Mux-180 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 6B Mux-186 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 68 Mux-187 Stuck-at-0
Input: 9893DC4C80 Circuit Output: 6A True Output: 68 Mux-192 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 6E Mux-194 Stuck-at-1
Input: 9893DC4C80 Circuit Output: 6A True Output: 68 Mux-195 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 61 Mux-1 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 7D Mux-15 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 7A Mux-24 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-28 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-30 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-31 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 70 Mux-49 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-51 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-52 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-64 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-78 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 61 Mux-92 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 70 Mux-93 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-95 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 6A Mux-101 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 60 Mux-102 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 5D Mux-112 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 70 Mux-138 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 6D Mux-145 Stuck-at-1

Input: C96F63C4E0 Circuit Output: 69 True Output: 70 Mux-148 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 60 Mux-151 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 60 Mux-172 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 6A Mux-180 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 60 Mux-181 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 6B Mux-187 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 68 Mux-188 Stuck-at-0
Input: C96F63C4E0 Circuit Output: 69 True Output: 68 Mux-193 Stuck-at-1
Input: C96F63C4E0 Circuit Output: 69 True Output: 68 Mux-196 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 1E Mux-4 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 1E Mux-6 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 5E Mux-8 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 58 Mux-32 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 1E Mux-39 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 50 Mux-41 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 50 Mux-53 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 1E Mux-58 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 50 Mux-88 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 40 Mux-94 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 50 Mux-97 Stuck-at-1
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 40 Mux-141 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 40 Mux-166 Stuck-at-0
Input: EFF7BBEBE0 Circuit Output: 4F True Output: 40 Mux-175 Stuck-at-1
Input: 1B3D624BB0 Circuit Output: 6A True Output: 70 Mux-43 Stuck-at-1
Input: 1B3D624BB0 Circuit Output: 6A True Output: 3D Mux-49 Stuck-at-0
Input: 1B3D624BB0 Circuit Output: 6A True Output: 4A Mux-50 Stuck-at-1
Input: 1B3D624BB0 Circuit Output: 6A True Output: 3D Mux-63 Stuck-at-1
Input: 1B3D624BB0 Circuit Output: 6A True Output: 4A Mux-76 Stuck-at-0
Input: 1B3D624BB0 Circuit Output: 6A True Output: 70 Mux-89 Stuck-at-1
Input: 1B3D624BB0 Circuit Output: 6A True Output: 4A Mux-93 Stuck-at-0
Input: 1B3D624BB0 Circuit Output: 6A True Output: 4A Mux-111 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-20 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-22 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-23 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 70 Mux-37 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-47 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-48 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-62 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-74 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 70 Mux-86 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-91 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 60 Mux-100 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 5C Mux-110 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 70 Mux-140 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 6C Mux-147 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 60 Mux-149 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 60 Mux-170 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 60 Mux-179 Stuck-at-1
Input: D76065A1C0 Circuit Output: 6B True Output: 68 Mux-186 Stuck-at-0
Input: D76065A1C0 Circuit Output: 6B True Output: 68 Mux-191 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-1 Stuck-at-1

Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-2 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 70 Mux-3 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-5 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-8 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-10 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 7E Mux-11 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-13 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-92 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-96 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-139 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-143 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-165 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-167 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-174 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 50 Mux-176 Stuck-at-1
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-183 Stuck-at-0
Input: 62F1F1FF60 Circuit Output: 56 True Output: 5E Mux-189 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 6F Mux-4 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-32 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-34 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-35 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 78 Mux-36 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-53 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-54 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-65 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-80 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 6F Mux-94 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-97 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 60 Mux-103 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 5B Mux-113 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 70 Mux-142 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 6B Mux-149 Stuck-at-1
Input: 34BE261250 Circuit Output: 68 True Output: 60 Mux-152 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 60 Mux-173 Stuck-at-0
Input: 34BE261250 Circuit Output: 68 True Output: 60 Mux-182 Stuck-at-1
Input: D142B516B0 Circuit Output: 36 True Output: 3E Mux-3 Stuck-at-1
Input: D142B516B0 Circuit Output: 36 True Output: 33 Mux-11 Stuck-at-1
Input: D142B516B0 Circuit Output: 36 True Output: 5C Mux-18 Stuck-at-1
Input: D142B516B0 Circuit Output: 36 True Output: 49 Mux-28 Stuck-at-0
Input: D142B516B0 Circuit Output: 36 True Output: 40 Mux-51 Stuck-at-1
Input: D142B516B0 Circuit Output: 36 True Output: 06 Mux-83 Stuck-at-1
Input: 3800B89690 Circuit Output: 09 True Output: 20 Mux-44 Stuck-at-0
Input: 3800B89690 Circuit Output: 09 True Output: 20 Mux-48 Stuck-at-0
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-9 Stuck-at-1
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-40 Stuck-at-1
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-67 Stuck-at-0
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-87 Stuck-at-0
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-114 Stuck-at-1
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-124 Stuck-at-0
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-136 Stuck-at-0
Input: 0E170FA880 Circuit Output: 5F True Output: 4F Mux-154 Stuck-at-1

Input: 8B6BB29E70 Circuit Output: 48 True Output: 50 Mux-95 Stuck-at-1
Input: 8B6BB29E70 Circuit Output: 48 True Output: 4E Mux-96 Stuck-at-1
Input: 8B6BB29E70 Circuit Output: 48 True Output: 49 Mux-102 Stuck-at-1
Input: 9A7782BD40 Circuit Output: 5E True Output: 39 Mux-41 Stuck-at-0
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-42 Stuck-at-1
Input: 9A7782BD40 Circuit Output: 5E True Output: 39 Mux-59 Stuck-at-1
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-69 Stuck-at-0
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-88 Stuck-at-0
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-115 Stuck-at-1
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-125 Stuck-at-0
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-137 Stuck-at-0
Input: 9A7782BD40 Circuit Output: 5E True Output: 4E Mux-155 Stuck-at-1
Input: DC93F1E130 Circuit Output: 27 True Output: 30 Mux-5 Stuck-at-0
Input: DC93F1E130 Circuit Output: 27 True Output: 20 Mux-7 Stuck-at-1
Input: DC93F1E130 Circuit Output: 27 True Output: 3A Mux-27 Stuck-at-0
Input: DC93F1E130 Circuit Output: 27 True Output: 60 Mux-39 Stuck-at-1
Input: DC93F1E130 Circuit Output: 27 True Output: 30 Mux-144 Stuck-at-1
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-33 Stuck-at-1
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-79 Stuck-at-0
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-120 Stuck-at-1
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-130 Stuck-at-0
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-146 Stuck-at-0
Input: DF092AAE00 Circuit Output: 59 True Output: 49 Mux-160 Stuck-at-1
Input: 0E4FE2A830 Circuit Output: 6E True Output: 7E Mux-13 Stuck-at-0
Input: 0E4FE2A830 Circuit Output: 6E True Output: 5F Mux-68 Stuck-at-0
Input: 0E4FE2A830 Circuit Output: 6E True Output: 5F Mux-107 Stuck-at-1
Input: 0E4FE2A830 Circuit Output: 6E True Output: 70 Mux-136 Stuck-at-1
Input: 0E4FE2A830 Circuit Output: 6E True Output: 6F Mux-141 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-14 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-15 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-17 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-44 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-70 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-71 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-89 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 30 Mux-98 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-108 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-116 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-126 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-138 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 30 Mux-145 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 2E Mux-156 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 3E Mux-167 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 30 Mux-168 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 30 Mux-177 Stuck-at-1
Input: B75D783480 Circuit Output: 3D True Output: 38 Mux-185 Stuck-at-0
Input: B75D783480 Circuit Output: 3D True Output: 38 Mux-190 Stuck-at-1
Input: 7ECE6F64E0 Circuit Output: 6E True Output: 66 Mux-139 Stuck-at-1
Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-25 Stuck-at-1
Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-75 Stuck-at-0
Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-118 Stuck-at-1

Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-128 Stuck-at-0
Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-142 Stuck-at-0
Input: B913341E80 Circuit Output: 3B True Output: 2B Mux-158 Stuck-at-1
Input: F2B9B15980 Circuit Output: 2A True Output: 30 Mux-135 Stuck-at-1
Input: 8BEC6254D0 Circuit Output: 7D True Output: 7A Mux-12 Stuck-at-1
Input: DAAA15F520 Circuit Output: 39 True Output: 3A Mux-150 Stuck-at-1
Input: DAAA15F520 Circuit Output: 39 True Output: 31 Mux-165 Stuck-at-1
Input: DAAA15F520 Circuit Output: 39 True Output: 3B Mux-170 Stuck-at-1
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-36 Stuck-at-1
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-81 Stuck-at-0
Input: 3B595186E0 Circuit Output: 58 True Output: 70 Mux-90 Stuck-at-1
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-121 Stuck-at-1
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-131 Stuck-at-0
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-148 Stuck-at-0
Input: 3B595186E0 Circuit Output: 58 True Output: 49 Mux-161 Stuck-at-1
Input: 3B595186E0 Circuit Output: 58 True Output: 59 Mux-172 Stuck-at-1
Input: 032BC9FDC0 Circuit Output: 38 True Output: 3C Mux-169 Stuck-at-1
Input: 1773FBCEB0 Circuit Output: 49 True Output: 50 Mux-87 Stuck-at-1
Input: 9D897898A0 Circuit Output: 2F True Output: 0F Mux-66 Stuck-at-0
Input: 9D897898A0 Circuit Output: 2F True Output: 0F Mux-106 Stuck-at-1
Input: 711931B200 Circuit Output: 50 True Output: 10 Mux-37 Stuck-at-0
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-38 Stuck-at-1
Input: 711931B200 Circuit Output: 50 True Output: 10 Mux-55 Stuck-at-1
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-57 Stuck-at-0
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-86 Stuck-at-0
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-105 Stuck-at-1
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-123 Stuck-at-0
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-135 Stuck-at-0
Input: 711931B200 Circuit Output: 50 True Output: 40 Mux-153 Stuck-at-1
Input: 194C9603D0 Circuit Output: 6D True Output: 7D Mux-17 Stuck-at-0
Input: 194C9603D0 Circuit Output: 6D True Output: 2E Mux-43 Stuck-at-0
Input: 194C9603D0 Circuit Output: 6D True Output: 2E Mux-60 Stuck-at-1
Input: 65BBE85CC0 Circuit Output: 78 True Output: 7A Mux-171 Stuck-at-1
Input: D0BB01FFF0 Circuit Output: 30 True Output: 10 Mux-56 Stuck-at-0
Input: D0BB01FFF0 Circuit Output: 30 True Output: 10 Mux-104 Stuck-at-1
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-29 Stuck-at-1
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-77 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-119 Stuck-at-1
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-129 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-144 Stuck-at-0
Input: CFD8384190 Circuit Output: 7A True Output: 6A Mux-159 Stuck-at-1
Input: 3A65A60720 Circuit Output: 6D True Output: 70 Mux-137 Stuck-at-1
Input: 3A65A60720 Circuit Output: 6D True Output: 6E Mux-143 Stuck-at-1
Input: 0000000000 Circuit Output: 00 True Output: 30 Mux-46 Stuck-at-0
Input: 0000000000 Circuit Output: 00 True Output: 08 Mux-182 Stuck-at-0
Input: 0000000000 Circuit Output: 00 True Output: 08 Mux-184 Stuck-at-0

APPENDIX C

The following is the localization output results for pseudorandom sequential and parallel testing of the C432 benchmark circuit (the test patterns that yielded zero fault detection were omitted):

Input: 183A183AA0 Circuit Output: 00 True Output: 60 Mux-2 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 3F Mux-6 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 4E Mux-10 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 1D Mux-14 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 6C Mux-18 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 3B Mux-22 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 4A Mux-26 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 09 Mux-30 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 58 Mux-34 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-38 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 30 Mux-40 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-44 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-46 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 30 Mux-48 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-54 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-55 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-56 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-57 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-58 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-59 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-60 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-61 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-62 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-63 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-64 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-65 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-66 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-67 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-68 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-69 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-70 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-71 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-72 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-73 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-74 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-75 Stuck-at-1

Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-76 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-77 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-78 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-79 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-80 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-81 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-82 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 40 Mux-85 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-104 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-105 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-106 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-107 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-108 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-109 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-110 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-111 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-112 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-113 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-114 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-115 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-116 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-117 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-118 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-119 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-120 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-121 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-122 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-123 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-124 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-125 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-126 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-127 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-128 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-129 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-130 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-131 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 20 Mux-134 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-153 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-154 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-155 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-156 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-157 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-158 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-159 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-160 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-161 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-162 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 10 Mux-164 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 0F Mux-175 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 0E Mux-176 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 0D Mux-177 Stuck-at-0

Input: 183A183AA0 Circuit Output: 00 True Output: 0C Mux-178 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 0B Mux-179 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 0A Mux-180 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 09 Mux-181 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 08 Mux-182 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 08 Mux-184 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 05 Mux-185 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 03 Mux-186 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 02 Mux-187 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 01 Mux-188 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 08 Mux-189 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 05 Mux-190 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 03 Mux-191 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 02 Mux-192 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 01 Mux-193 Stuck-at-0
Input: 183A183AA0 Circuit Output: 00 True Output: 04 Mux-194 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 02 Mux-195 Stuck-at-1
Input: 183A183AA0 Circuit Output: 00 True Output: 01 Mux-196 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 6E Mux-11 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5D Mux-12 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5E Mux-13 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 6A Mux-27 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5A Mux-29 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-32 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-34 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 78 Mux-35 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-36 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-53 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-54 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-65 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-81 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 1D Mux-82 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5D Mux-84 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 18 Mux-85 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-97 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5D Mux-98 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-103 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-121 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-131 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-132 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-133 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-148 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-152 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-161 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 4E Mux-162 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5E Mux-163 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 48 Mux-164 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5E Mux-167 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 5A Mux-171 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-173 Stuck-at-0
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-174 Stuck-at-0

Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-182 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-183 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-184 Stuck-at-1
Input: 18FE18FEE0 Circuit Output: 58 True Output: 50 Mux-189 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 49 Mux-12 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 49 Mux-14 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 7D Mux-15 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-17 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 69 Mux-31 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-43 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-44 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-60 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-71 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-83 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-89 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 50 Mux-98 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-116 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-126 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-138 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 50 Mux-145 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 4D Mux-156 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 50 Mux-168 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 50 Mux-177 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 58 Mux-185 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 58 Mux-190 Stuck-at-1
Input: 016F016FF0 Circuit Output: 5D True Output: 59 Mux-194 Stuck-at-0
Input: 016F016FF0 Circuit Output: 5D True Output: 5C Mux-196 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 55 Mux-1 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-16 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-18 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 7C Mux-19 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-21 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-28 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-30 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-33 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 55 Mux-92 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-99 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-102 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-147 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-151 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-169 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-172 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 59 Mux-178 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-181 Stuck-at-1
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-188 Stuck-at-0
Input: E326E32660 Circuit Output: 5D True Output: 5C Mux-193 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 7F Mux-7 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-19 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-45 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-46 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-61 Stuck-at-1

Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-72 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 6C Mux-73 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-90 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-109 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 6C Mux-117 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 5F Mux-122 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 6C Mux-127 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 7F Mux-133 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 5C Mux-134 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 6C Mux-140 Stuck-at-0
Input: C712C71220 Circuit Output: 7C True Output: 7F Mux-141 Stuck-at-1
Input: C712C71220 Circuit Output: 7C True Output: 6C Mux-157 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 5F Mux-4 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-6 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-9 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 5B Mux-20 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 3B Mux-23 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 50 Mux-39 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 50 Mux-47 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 20 Mux-52 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 0F Mux-83 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 10 Mux-84 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-94 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-141 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-166 Stuck-at-0
Input: 3639363990 Circuit Output: 1F True Output: 1B Mux-175 Stuck-at-1
Input: 3639363990 Circuit Output: 1F True Output: 1D Mux-195 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 2E Mux-1 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 2E Mux-2 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 60 Mux-3 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 50 Mux-5 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 6E Mux-8 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 44 Mux-16 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 6A Mux-24 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 2E Mux-37 Stuck-at-0
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 60 Mux-41 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 60 Mux-49 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 2E Mux-55 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 60 Mux-88 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 60 Mux-93 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 47 Mux-94 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 46 Mux-96 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 44 Mux-99 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 43 Mux-100 Stuck-at-1
Input: 7FDB7FDBB0 Circuit Output: 40 True Output: 42 Mux-101 Stuck-at-1
Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-5 Stuck-at-1
Input: E3B1E3B110 Circuit Output: 14 True Output: 50 Mux-37 Stuck-at-1
Input: E3B1E3B110 Circuit Output: 14 True Output: 50 Mux-45 Stuck-at-1
Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-92 Stuck-at-0
Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-139 Stuck-at-0
Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-165 Stuck-at-0

Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-174 Stuck-at-1
Input: E3B1E3B110 Circuit Output: 14 True Output: 1C Mux-183 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-8 Stuck-at-1
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-10 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-11 Stuck-at-1
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-96 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 70 Mux-132 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 70 Mux-136 Stuck-at-1
Input: 264B264BB0 Circuit Output: 6E True Output: 70 Mux-142 Stuck-at-1
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-143 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 60 Mux-163 Stuck-at-1
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-167 Stuck-at-0
Input: 264B264BB0 Circuit Output: 6E True Output: 6A Mux-176 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 43 Mux-4 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 5F Mux-9 Stuck-at-0
Input: 6F956F9550 Circuit Output: 47 True Output: 5B Mux-25 Stuck-at-0
Input: 6F956F9550 Circuit Output: 47 True Output: 79 Mux-28 Stuck-at-0
Input: 6F956F9550 Circuit Output: 47 True Output: 68 Mux-32 Stuck-at-0
Input: 6F956F9550 Circuit Output: 47 True Output: 70 Mux-43 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 70 Mux-51 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 60 Mux-53 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 70 Mux-89 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 70 Mux-95 Stuck-at-1
Input: 6F956F9550 Circuit Output: 47 True Output: 60 Mux-97 Stuck-at-1
Input: F8BCF8BCC0 Circuit Output: 78 True Output: 58 Mux-35 Stuck-at-1
Input: F8BCF8BCC0 Circuit Output: 78 True Output: 58 Mux-80 Stuck-at-0
Input: F8BCF8BCC0 Circuit Output: 78 True Output: 58 Mux-113 Stuck-at-1
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-38 Stuck-at-1
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-57 Stuck-at-0
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-86 Stuck-at-0
Input: 63B763B770 Circuit Output: 50 True Output: 51 Mux-102 Stuck-at-1
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-105 Stuck-at-1
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-123 Stuck-at-0
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-135 Stuck-at-0
Input: 63B763B770 Circuit Output: 50 True Output: 40 Mux-153 Stuck-at-1
Input: 80F280F220 Circuit Output: 5E True Output: 5A Mux-13 Stuck-at-1
Input: B802B80220 Circuit Output: 00 True Output: 30 Mux-42 Stuck-at-0
Input: B802B80220 Circuit Output: 00 True Output: 30 Mux-50 Stuck-at-0
Input: 46CF46CFF0 Circuit Output: 70 True Output: 6E Mux-3 Stuck-at-1
Input: 46CF46CFF0 Circuit Output: 70 True Output: 66 Mux-56 Stuck-at-0
Input: 46CF46CFF0 Circuit Output: 70 True Output: 66 Mux-104 Stuck-at-1
Input: 46CF46CFF0 Circuit Output: 70 True Output: 75 Mux-145 Stuck-at-1
Input: B806B80660 Circuit Output: 59 True Output: 5D Mux-17 Stuck-at-0
Input: B806B80660 Circuit Output: 59 True Output: 5D Mux-168 Stuck-at-1
Input: A3FDA3FDD0 Circuit Output: 2D True Output: 29 Mux-15 Stuck-at-1
Input: A3FDA3FDD0 Circuit Output: 2D True Output: 39 Mux-33 Stuck-at-0
Input: A3FDA3FDD0 Circuit Output: 2D True Output: 38 Mux-36 Stuck-at-0
Input: A3FDA3FDD0 Circuit Output: 2D True Output: 2E Mux-143 Stuck-at-1
Input: DFC6DFC660 Circuit Output: 6C True Output: 7C Mux-21 Stuck-at-0
Input: DFC6DFC660 Circuit Output: 6C True Output: 70 Mux-146 Stuck-at-1
Input: DFC6DFC660 Circuit Output: 6C True Output: 70 Mux-148 Stuck-at-1

Input: DFC6DFC660 Circuit Output: 6C True Output: 6D Mux-151 Stuck-at-1
Input: 8DB68DB660 Circuit Output: 6F True Output: 6B Mux-7 Stuck-at-1
Input: 1C4A1C4AA0 Circuit Output: 6E True Output: 70 Mux-87 Stuck-at-1
Input: 1C4A1C4AA0 Circuit Output: 6E True Output: 70 Mux-91 Stuck-at-1
Input: 108D108DD0 Circuit Output: 7D True Output: 6D Mux-70 Stuck-at-0
Input: 108D108DD0 Circuit Output: 7D True Output: 6D Mux-108 Stuck-at-1
Input: F774F77440 Circuit Output: 78 True Output: 7C Mux-147 Stuck-at-1
Input: 4ED74ED770 Circuit Output: 7E True Output: 76 Mux-165 Stuck-at-1
Input: 3874387440 Circuit Output: 78 True Output: 7A Mux-150 Stuck-at-1
Input: E5DCE5DCC0 Circuit Output: 78 True Output: 7F Mux-166 Stuck-at-1
Input: E5DCE5DCC0 Circuit Output: 78 True Output: 7B Mux-170 Stuck-at-1
Input: C2F5C2F550 Circuit Output: 5E True Output: 70 Mux-86 Stuck-at-1
Input: C2F5C2F550 Circuit Output: 5E True Output: 70 Mux-90 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-31 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-51 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-52 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-64 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-78 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-79 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-95 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-112 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-120 Stuck-at-1
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-130 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-146 Stuck-at-0
Input: 8005800550 Circuit Output: 79 True Output: 6D Mux-160 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 70 Mux-137 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 70 Mux-144 Stuck-at-1
Input: 4EF14EF110 Circuit Output: 60 True Output: 63 Mux-149 Stuck-at-1
Input: 88EC88ECC0 Circuit Output: 78 True Output: 79 Mux-172 Stuck-at-1
Input: 4CAF4CAFF0 Circuit Output: 67 True Output: 70 Mux-138 Stuck-at-1
Input: 79B679B660 Circuit Output: 58 True Output: 5C Mux-169 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 30 Mux-135 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 26 Mux-139 Stuck-at-1
Input: E3DFE3DFF0 Circuit Output: 2E True Output: 30 Mux-140 Stuck-at-1