# Serial Fault Emulation

Luc Burgun, Frédéric Reblewski, Gérard Fenelon,
Jean Barbier and Olivier Lepape
META SYSTEMS*
4, Rue René Razel, 91400 Saclay France

ABSTRACT - *A hardware emulator based approach has been developed to perform test evaluation on large sequential circuits (at least tens of thousands of gates). This approach relies both on the flexibility and on the reconfigurability of hardware emulators based on dedicated reprogrammable circuits. A Serial Fault Emulation (SFE) method in which each faulty circuit is emulated separately has been applied to gate-level circuits for Single Stuck Faults (SSFs).*

*This approach has been implemented on the Meta Systems's hardware emulator which is capable of emulating circuits of 1,000,000 gates at rates varying from 500KHz to several MHz. Experimental results are provided to demonstrate the efficiency of SFE. They indicate that SFE should be two orders of magnitude faster than software approaches for designs containing more than 100.000 gates.*

## 1 INTRODUCTION

Test evaluation consists in determining the effectiveness of a set of test patterns by computing the ratio between the number of faults detected by this set and the total number of possible faults with respect to a given fault model. The traditional approach to test evaluation relies on software programs simulating the effects of the faults on the behavior of the circuit. The simplest method, called *serial fault simulation* simulates the faulty circuits, one at a time. This method does not require a dedicated fault simulator (any logic simulator can be easily adapted). Therefore this method *a priori* can handle any type of fault [1]. However, due to the performances of software logic simulators, this type of fault simulation is completely impractical if a large number of faults has to be considered [1, 14].

In the last decades, more sophisticated general purpose methods have been proposed such as *parallel* [14], *concurrent* [15] or *differential fault simulation* [5]. These techniques differ from serial fault simulation because they aim at minimizing the number of simulation passes by simultaneously processing faults. The concurrent method is implemented in most commercial tools because of its generality and its efficiency [6].

Today, the fault simulation approach is becoming unrealistic for many designs not only because the theoretical complexity for simulating one pattern appears to be between linear and quadratic with the number of gates [8], but also because the complexity of the circuits increases faster than the computing speed [2].

Recently, a new approach based on reprogrammable hardware has been proposed [9, 17, 4] to verify circuits before committing them to silicon. This approach called *logic* or *hardware emulation* [4, 11] decreases the design time by allowing a "real-time" verification 10,000 to 1,000,000 times faster than software logic simulation [11].

In this paper, we propose a methodology to extend the utilization of hardware emulators for test evaluation by using a brute-force method, called *serial fault emulation*, in which the fault-free circuit and the faulty circuits are considered separately. Unlike hardware accelerators dedicated to logic simulation [16], a significant speed-up can be achieved in comparison with the *state-of-the-art* software fault simulation methods because of the performances of hardware emulators. The main advantage of SFE is that the run time is *quasi*-proportional to the number of faults so that test evaluation can be performed for very large circuits with large test sets.

A fast Computer-Aided Prototyping (CAP) software package combining netlist translation, synthesis, multi-chip partitioning and routing automatically produces a hardware prototype of the fault-free circuit. A partial reconfiguration of the hardware emulator is then computed for each fault of interest. The configuration file related to the hardware prototype is downloaded into the hardware emulator and a first emulation pass allows the verification or the calculation of the expected values of the test set. The faults are then emulated one at a time by partially modifying the fault-free hardware prototype so that it models each faulty circuit.

This paper deals with sequential circuits described at the gate level (gate netlist). As mentioned for serial fault simulation [1], SFE may be used for other types of faults such as multiple fault or bridging fault, but we will concentrate on the SSF model.

This paper is organized as follows. Section 2 briefly describes the CAP software of the *Meta Systems*'s hardware emulator. Section 3 presents our approach for fault emulation and shows particularly how to compute each faulty circuit from the fault-free circuit and the fault to be inserted. Section 4 deals with the problem of minimizing the run time for SFE and explains the techniques for limiting the software tasks. Section 5 presents experimental results and Section 6 concludes this paper.

## 2 CAP FOR LOGIC EMULATION

This chapter briefly describes the CAP software used for implementing circuits on the *Meta Systems*'s hardware emulator. As shown on the right part of Figure 1, the first step consists in translating the design netlist into the *Meta Systems* internal format, namely *ANF*. This format supports hierarchical descriptions and allows the use of any 4-input function cell. Each cell of the design library is mapped onto the *Meta Systems* library (called *metalib*). The resulting library (the *conversion library*) is used for each design to express the ANF gate netlist in terms of cells of the metalib.

The netlist is then flattened, optimized and targeted to the architecture of the reprogrammable circuits used in the hardware emulator, namely the *Metas*. The architecture of the *Meta* consists of a column of logic blocks and a global interconnexion matrix (crossbar) which connects the I/Os and the logic blocks (*BLPs*). The crossbar improves the inter-chip communication by removing the constraints related to I/O placement (each BLP may be connected to any I/O without decreasing the percentage of BLP utilization). Each BLP consists of a 4-input Sram and a reprogrammable sequential device which can emulate either a flip-flop or a latch.

As the *Tabula Rasa* chip [9], the *Meta* is targeted specifically for logic emulation. In contrast with the *Xilinx* LCA architec-

* META SYSTEMS is now part of MENTOR GRAPHICS CORPORATION

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA 

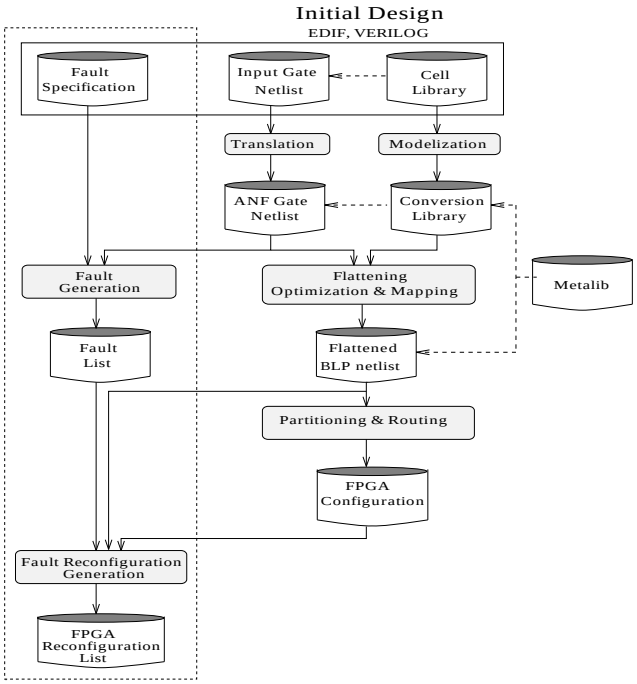


Fig. 1: Flowchart of the *Meta Systems*'s CAP software and the FPGA reconfiguration generation software (denoted by the dashed box)

ture [10], each BLP may be observed without adding routing constraints which may cause congested areas and consequently lead to routing failures. This important feature avoids the need to re-compile the design netlist when the user wishes to observe different signals from those initially declared as probes.

After targeting the netlist to our reprogrammable hardware architecture, the netlist is partitioned into two levels of hierarchy, namely *Metas* and boards. The partitioner implements efficient techniques such as *logic replication* for reducing the pin count and the partition size. Unlike Quickturn's RPM system [17], the partitioner does not make use of the designer's hierarchy.

Finally, the system achieves the routing at three level of interconnections corresponding to *Metas*, boards and backplane board. Each logic board consists of 3 processing columns separated by 2 routing columns. Each processing column contains 8 processing elements, each one consisting of a *Meta*, a 32K byte memory and a Video VRAM in which the values of all BLPs in the *Meta* for the last 7,200 emulation cycles are stored. The backplane board connects 23 logic boards and an interface board managing the communications between the hardware emulator and the workstation host. Several backplane boards (up to 6) may be linked to emulate very large designs.

The routing step produces a configuration file which is downloaded into the hardware emulator before operating the hardware prototype. The operating environment consists of a user-friendly interface in *Motif* and a C interpreter which allows the description of emulation experiences. An emulation experience defines the conditions in which the hardware prototype operates :

- Maximum number of cycles
- Maximum speed of operating
- Initial values for sequential devices (registers and memories)
- Stopping conditions

A stopping condition is defined by arming a hardware trigger which tests when the emulation brings one or more registers into a predefined state. Unlike the Quickturn's system [7], not only the triggers can be changed without re-compiling the prototype, but also every register can be used in a stopping condition.

## 3 Overview of the Fault Emulation System

Fault emulation involves calculating a reconfiguration of the hardware emulator for each fault of interest (for the sake of simplicity, we will use the term FPGA[1] reconfiguration).

For this purpose, we have developed specific tools which operate in parallel with the CAP software.

### 3.1 Calculation of FPGA Reconfigurations

The left side of Figure 1 indicates how the FPGA reconfigurations are computed with respect to the CAP software. A fault generator constructs the collapsed fault list from the *ANF* gate netlist and from a fault specification file. This file specifies the blocks of the hierarchy in which the faults will be inserted and the faults excluded from fault emulation.

A second step consists in calculating the FPGA reconfiguration associated with each fault of interest so that the modified hardware prototype behaves like the faulty circuit. To minimize the total run time for fault emulation, each FPGA reconfiguration has to affect as few BLPs as possible. Hence, in contrast with logic emulation, the CAP software has to be restricted so that it does not result in large modifications to the original netlist. This restriction excludes the use of re-synthesis techniques relying on logic level optimization techniques such as *extraction* or *substitution* [3]. Hence, the optimization and mapping phase consists only in collapsing the single fanout gates into nodes which satisfy the 4-input constraint. In these conditions, if a gate has a multiple fanout, the gate cannot be collapsed so that its output signal is kept in the BLP netlist. This comes down to separately map each fanout free region.

An FPGA reconfiguration corresponds to a list of BLPs to be reprogrammed in order to generate the faulty circuit from the fault-free circuit and *vice versa*. The cases where it is necessary to reconfigure more than one BLP are as follows :

- A primary input pin of the circuit has a multiple fanout
- An input pin of a gate of the design library has a multiple fanout in the equivalent cell of the conversion library
- A BLP is replicated during the partioning phase

In the two first cases, the pin stuck fault results in stucking all the input pins of the gates of the multiple fanout so that SSF has to be emulated by a multiple stuck-at fault.

Each BLP of the reconfiguration is associated with a logical address in the emulator and two words encoding the functionality of the BLP for the fault-free circuit and the faulty circuit. The logical address is a 3-uple denoting the board number in the machine, the *Meta* number in the board and the BLP number in the *Meta*.

### 3.2 Fault Insertion for Combinational Circuits

The FPGA reconfiguration for SSF on a combinational gate affects only the 4-input function of the BLP.

Consider the circuit in Figure 2 and suppose that the gates $G_0$, $G_1$ and $G_2$ are gathered into the BLP $(0, 0, 0)$ (denoting the board 0, *Meta* 0 and BLP 0) and the gate $G_3$ corresponds to the BLP $(0, 0, 1)$. The 4-input function of the BLP $(0, 0, 0)$ is $F = A.B + C.D$. If the signal $X$ is stuck at zero, this BLP has to be reconfigured so that it implements the function $F = C.D$.

[1]The term FPGA is used to refer to all types of field programmable logic, both LCAs such as Xilinx and also those more commonly refered to as PALs and PLDs

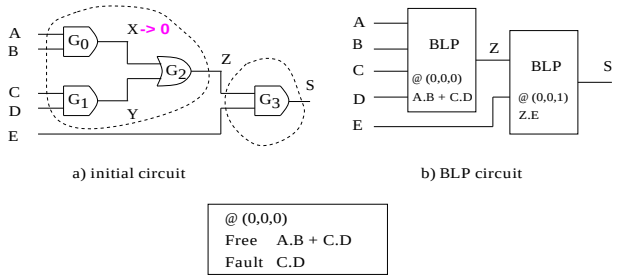


Fig. 2: An example of FPGA Reconfiguration

Note that partial fault collapsing may be easily achieved by identifying the faults which produce identical FPGA reconfigurations. In the example, the signals $A, B, X$ stuck-at-0 produce the same FPGA reconfiguration so that only one emulation run will be necessary to test these three faults.

### 3.3 Fault Insertion for Sequential Circuits

As mentioned in Section 2, each BLP consists of a 4-input Sram and a sequential device. This later may be configured as either an edge-triggered flip-flop or a latch and it may have additional features such as *load enable*, asynchronous *reset* and *set* lines. The sequential devices are synchronized by a complex clock system ensuring that there are no hold time violations due to short-pathes between registers.

The FPGA reconfiguration for SSF on a sequential gate affects both the combinational section and the sequential section of the BLP. Table 1 shows how a BLP emulating a flip-flop with *reset*, *set* and *enable* lines is reconfigured according to the stuck faults on its pins (assume active high on all signals).

| Fault | Type | F | RST | SET | EN | CLK |
|---|---|---|---|---|---|---|
| none | Seq | $D$ | + | + | + | + |
| D Sat0 | Seq | 0 | + | + | + | + |
| D Sat1 | Seq | 1 | + | + | + | + |
| Q Sat0 | Comb | 0 | - | - | - | - |
| Q Sat1 | Comb | 1 | - | - | - | - |
| RST Sat0 | Seq | $D$ | - | + | + | + |
| RST Sat1 | Comb | 0 | - | - | - | - |
| SET Sat0 | Seq | $D$ | + | - | + | + |
| SET Sat1 | Comb | $\overline{RST}$ | - | - | - | |
| EN Sat0 | Seq | - | + | + | - | - |
| EN Sat1 | Seq | $D$ | + | + | - | + |

Table 1: FPGA reconfiguration for SSF on a register

For each SSF of a register, this table shows whether the BLP remains a sequential device ($Seq$) or becomes a combinational gate ($Comb$), the new function $F$ , and whether the *reset*, *set*, *enable* and *clock* lines are used ($+$) or not ($-$). For example when *reset* is stuck at 1, the BLP becomes combinational and it implements the function $F = 0$.

### 3.4 Fault Emulation

Once both the basic configuration of the prototype and the fault reconfigurations have been generated, fault emulation can be performed. There are three emulation modes.

- Fault-free Circuit Emulation
- Faulty Circuit Emulation
- Serial Fault Emulation

The first mode is used to debug the fault-free circuit before running fault emulation. In certain cases, this mode may also be used to compute the expected values for the observed outputs.

Faulty circuit emulation allows the effects of a SFF to be observed. The high observability of the *Meta* makes it possible to know where an undetected fault is blocked for a given pattern. This feature is useful for test coverage improvements or for analyzing undetectable faults.

Serial fault emulation is the most important mode because it allows the calculation of the fault coverage and the construction of the fault dictionary. This mode runs through all faulty circuit emulations as fast as possible. A faulty circuit is processed following the five basic steps :

1. Fault insertion by reconfiguring the emulator
2. Register initialization
3. Trigger setting for the fault dropping
4. Faulty circuit emulation
5. Fault deletion by reconfiguring the emulator

During step 4, if an output value differs from the expected value, the trigger (set in step 3) is turned on and it activates the fault dropping. If that does not happen, the faulty circuit emulation continues until the test is finished.

In order to minimize the overhead for each fault processing, the FPGA reconfiguration has to be downloaded quickly. The problem of improving the fault emulation speed is addressed in the following section.

## 4 Improving the Fault Emulation Speed

The fault emulation speed is defined as the maximal number of faults processed per seconds. This speed basically depends on four factors :

- The average emulation runtime for processing a fault
- The time required to reconfigure the emulator
- The time required to initialize the sequential devices
- The time required to perform fault detection

Obviously, as the test set becomes larger, the emulation runtime becomes more significant. Conversely the three last factors are crucial when the test set is not very large or when most of faults are detected during the first cycles of the emulation. We will see in Section 4.4 and 4.5 that register initialization and fault detection may be performed by hardware so that the fault emulation speed depends only on the first two factors.

### 4.1 Emulation Runtime

The emulation runtime depends both on the number of cycles to be executed and on the maximal operating frequency of the prototype, namely the maximum clock speed ($MCS$). This frequency is computed by a worst-case static timing analyzer from the fault-free circuit. The static analysis ensures that each faulty circuit runs properly even if an inserted fault causes new dynamic pathes to occur.

Assume that $P$ is the average number of patterns necessary to detect the faults. If we neglect the overhead for each fault processing, the serial fault emulation speed ($S_{SFE}$) is expressed as follows :

$$S_{SFE} = \frac{MCS}{P}$$

Depending on the design, $MCS$ typically varies from 500 KHz to 5 MHz. Assuming that a given circuit operates at 1MHz and that the faults are detected in average at $P$ = 10,000, then the emulator will be able to process 100 faults per second.

### 4.2 Fast Reconfiguration

Let $T_{reconf}$ be the time required to reconfigure the hardware prototype, the SFE speed is now defined as follows :

$$S_{SFE} = \frac{MCS}{P + T_{reconf}.MCS}$$

$T_{reconf}$ depends mainly on the time needed to reconfigure BLPs. Unlike *Xilinx*, the *Meta* architecture provides the ability to read or modify only a portion of the chip at a time. This feature allows the reconfiguration time to be significantly reduced. A BLP can be reconfigured in 0.2 millisecond regardless of its location.

On average, $T_{reconf}$ is equal to 0.8 millisecond (4 BLPs to be reconfigured) and consequently 1,200 faults can be processed every second if $P$ is close to 0 (all the faults are detected in the first cycles). Assuming that a circuit operates at 1 Mhz, then the time required to reconfigure the prototype will be greater than the emulation runtime if $P$ < 800. Conversely the reconfiguration time will be negligible as soon as $P$ > 10,000.

### 4.3 Theoretical Complexity

It is clear that $S_{SFE}$ depends on the size of the circuits and that the run time cannot be considered as linear with the number of gates. On the other hand, there exists no explicit relation between the number of gates and the maximum clock frequency. Furthermore experimental results show that some large circuits can operate at higher speed than small circuits. Hence, SFE can be considered as *quasi*-linear with the number of gates of the circuit.

### 4.4 Register Initialization

Test evaluation generally requires the capability of bringing the circuits in a given state without applying any initialization sequence. This may be used when the test sets are so large that they have to be cut in several test sequences.

This may also be used in the following case. When hardware reset is not available on the registers, the test set consists of an initialization sequence which brings the circuit in a given state followed by an actual test sequence. In this case, register initialization may also be used to separately observe the effects of faults for the initializing sequence and the actual test sequence.

In logic emulation, the time required to initialize the registers is not significant in comparison with the emulation runtime. As indicated in Section 2, the registers (and more generally the sequential devices) may be initialized by writing a C program in which specified values (0 or 1) are assigned to them. When the program is interpreted, the BLP corresponding to each register is forced to the specified value. Note that the registers which are not forced by the program remain in an unpredictable state (either 0 or 1). The main drawback of this "software" initialization technique is that the time required to force the BLPs can significantly increase the overhead between each fault processing.

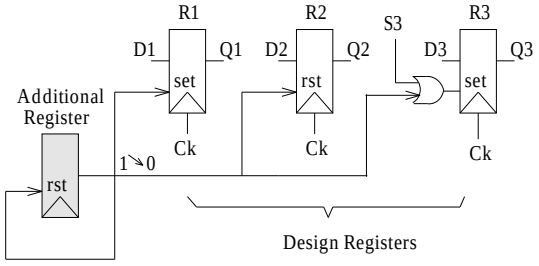


Fig. 3: Hardware initialization of Registers

We propose a hardware technique for initializing the registers into a given state. This technique takes advantage of the capability of forcing the value of the sequential device of each BLP with the asynchronous *set* and *reset* lines. In Figure 3, the register R1 and R3 have to be forced to 1 whereas the register R2 has to be forced to 0. An additional register is connected to the *set* or *reset* pins of the registers to be initialized. If a pin is already connected to another gate, a 2-input OR gate is added to the design for ORing the initial signal and the forcing line (R3 for example). In this case, the *set* or *reset* pin stuck faults are inserted on the input pins of the OR gate. At the beginning of the faulty circuit emulation, the additional register is forced to 1 by the software initialization described above so that it brings the circuit in the given state.

Note that if the user wants to bring the circuit in another initial state, the hardware prototype has to be re-compiled by the CAP software.

Unlike software initialization, hardware initialization avoids spending time between each fault pass. Furthermore, this technique has a negligible impact on the initial netlist and consequently on $MCS$.

### 4.5 Fault Detection

The *Meta Systems* hardware emulator provides two techniques for injecting stimuli. The first consists in using the 24 hardware memories available on each logic board in a generator mode. In this mode, each memory is directly addressed by a non-reconfigurable hardware counter which has to be reset at the beginning of each emulation. The user can load up to 32K patterns regardless of the number of bits. It is possible to extend the number of patterns capacity by successively loading 32K pattern pages. However this last possibility is not well-suited for fault emulation because of the loading time.

The second technique consists in replacing one or more logic boards by specialized memory boards. Each memory board can also operate in a generator mode and it can be loaded with up to 256K patterns of 384 bits. Memory boards may be combined to provide a very large memory.

To improve the fault detection speed, both the controlled input values and the expected output values are considered as stimuli. Hence, it is possible to perform a hardware comparison between the output values calculated by emulation and the expected values.

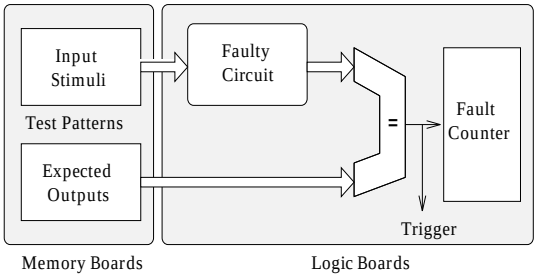


Fig. 4: Hardware Fault Detection

As shown in Figure 4, an equality comparator is inserted into the initial design so that only one signal has to be tested to verify whether the output values are different from the expected values or not. The trigger defined in Section 3.4 is set on this signal.

A counter may also be inserted into the design in order to calculate the number of times a fault is detected. In this mode, the trigger is turned off to prevent the emulation of the faulty circuit from being stopped before the end of the test.

The comparator may have an impact on $MCS$ since an observed signal can be located on the critical path (calculated by the static timing analyzer). In this case, the propagation time through the comparator is added to the critical path so that $MCS$ decreases. Each observed signal is propagated within the comparator through a 2-input NXOR and a N-input AND (where $N$ is

the number of observed signals). If a balanced technique is used for the mapping of the N-input AND, there are $1 + log_4(N)$ BLPs between each observed signal and the output of the comparator on which the trigger condition is set. Furthermore the comparator induces a partitioning and routing constraint since the observed signals have to be connected to the comparator through one or several *Metas* and/or boards.

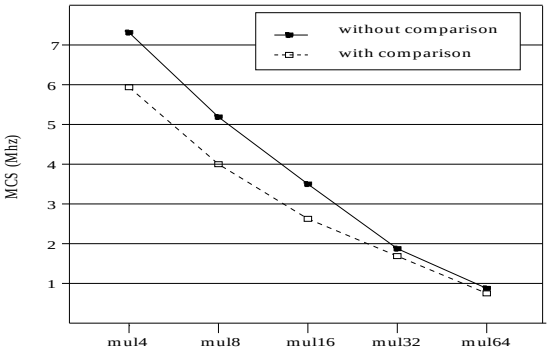


Fig. 5: Impact of the hardware comparison on MCS

Figure 5 shows the impact of the hardware comparison on $MCS$. We have selected several Booth's multipliers varying from 4 to 64 inputs. Each multiplier is generated with the CAP software and $MCS$ is calculated with the hardware comparator (denoted by the dashed line) and without the hardware comparator (denoted by the solid line). It can be seen that the propagation time through the comparator is added to the critical path of each multiplier since those circuits are combinational and all the outputs are observed (and compared). The two curves show that as the circuit size increases, the effect of the comparator decreases. Since our approach is targeted for large circuits, the effects of the hardware comparison can be disregarded. Note that for smaller circuits, the impact can nevertheless be minimized by using a pipeline architecture in which registers are inserted after the observed signals.

## 5 EXPERIMENTAL RESULTS

In order to measure the efficiency of SFE, we have conducted two sets of experiments.

### 5.1 Evolution of performances for a fixed architecture

First, we have conducted experiments to demonstrate the range of performance according to the size of a fixed architecture, namely for several Booth's multipliers made up of 2-input gates. We have generated a 1K random test patterns (this size is sufficient to obtain a 95% coverage). Each I/O pin of the gates of the circuits are stuck at 0 and 1, but fault collapsing is performed to minimize the number of faulty circuit runs. Table 2 indicates the number of gates (*#G*), the time required to compile the fault-free hardware prototype and to compute the FPGA reconfigurations ($T_{CAP}$) on a Sun workstation (Sparc 10 - 64 MBytes RAM), the number of faults (*#F*), the number of runs needed to test all the faults (*#R*) and the runtime for fault emulation ($T_{SFE}$).

Most of the faults (90%) are detected in the first ten patterns so that fault emulation runs at the maximal reconfiguration speed (around 1,200 runs per second). It is obvious that in these conditions $T_{SFE}$ is linear with the number of gates #$G$.

The time required to CAP is considerably greater than the runtime for fault emulation (this is especially true when the circuits are large). However it is not necessary to re-compile the circuits for other test sets or other fault sets.

### 5.2 SFE on ISCAS'89 Benchmarks

In the second experiment, we have performed test evaluation with 50K random test patterns on the largest sequential circuits from the standard set of the ISCAS'89 benchmarks. We have considered the SSF model for all the gate outputs without collapsing the faults. A full scan path has been inserted into the original design so that the random test can bring the circuit into many distinct states to obtain a good fault coverage (obviously, our method does not intrinsically require full scanned circuits). Furthermore, all the flip-flops are set to 0 at the beginning of each faulty circuit emulation by using the technique explained in Section 4.4.

| | CAP | | SFE | | |
|---|---|---|---|---|---|
| Circuit | #G | $T_{CAP}$ [sec.] | #F | #R | $T_{SFE}$ [sec.] |
| mul4x4 | 179 | 1.9 | 1000 | 424 | 0.3 |
| mul8x8 | 699 | 4.6 | 3876 | 1628 | 1.3 |
| mul16x16 | 2561 | 14.2 | 14620 | 6100 | 5.1 |
| mul32x32 | 10203 | 137.3 | 57320 | 23300 | 19.6 |
| mul64x64 | 39899 | 1950.2 | 218860 | 88962 | 77.4 |

Table 2: Fault emulation of multipliers

| | Description | | | | CAP | |
|---|---|---|---|---|---|---|
| Circuit | #G | #FF | #I | # O | $T_{CAP}$ [sec.] | $MCS$ [MHz] |
| s9234 | 5725 | 228 | 19 | 22 | 48.5 | 1.3 |
| s13207 | 8620 | 669 | 31 | 121 | 91.1 | 1.2 |
| s15850 | 10369 | 597 | 14 | 87 | 88.7 | 1.3 |
| s35932 | 17793 | 1728 | 35 | 320 | 279.2 | 2.1 |
| s38584 | 19705 | 1452 | 12 | 278 | 385.3 | 1.1 |
| s38417 | 23715 | 1636 | 28 | 106 | 356.2 | 1.5 |

Table 3: CAP for ISCAS'89 benchmarks

Table 3 gives the description of these circuits in terms of the number of gates (*G*), the number of flip-flops (*FF*), the number of inputs (*I*) and the number of outputs (*O*). The CAP results are reported in terms of the time required to compile the fault-free hardware prototype and to compute the FPGA reconfigurations ($T_{CAP}$) on a Sun workstation (Sparc 10 - 64 MBytes RAM) and the maximum clock speed ($MCS$).

Table 4 reports the results obtained by SFE in terms of the number of faults (#F), the fault coverage ($C$) obtained with the random test set, the average number of patterns needed to detect a fault ($T$), the runtime for fault emulation ($T_{SFE}$) and the fault emulation speed ($S_{SFE}$).

| | SFE | | | | |
|---|---|---|---|---|---|
| Circuit | #F | $C$ | $T$ | $T_{SFE}$ [sec.] | $S_{SFE}$ [f./sec.] |
| s9234 | 13020 | 81.1 | 15432 | 148.1 | 87.9 |
| s13207 | 21256 | 82.7 | 11501 | 196.4 | 108.2 |
| s15850 | 24322 | 82.6 | 12611 | 231.3 | 105.1 |
| s35932 | 45956 | 91.6 | 9781 | 169.7 | 270.8 |
| s38584 | 50124 | 92.7 | 4946 | 225.9 | 221.8 |
| s38417 | 57448 | 95.1 | 4531 | 170.5 | 336.9 |

Table 4: Results for 50K Pattern Fault Emulation

$MCS$ does not decrease with the complexity of the circuits so that SFE is linear in time with the number of gates. $S_{SFE}$ increases as the average number of patterns needed to detect a fault decreases.

We have compared our results with those obtained by *HOPE* (version 1.1) [12] [13] , a *state-of-the-art* fault simulator combining efficient techniques such as *single fault propagation* and

*parallel fault processing*. *HOPE* has been tested under similar conditions using the same fault model and the same test set. Furthermore, the performance of *HOPE* is measured on the same machine (Sparc 10). To compare the evolution of performance with the number of gates, we have to normalize the run time according to the average number of patterns required to detect a fault. So we have normalized the results according to the first circuit ($s9234$).

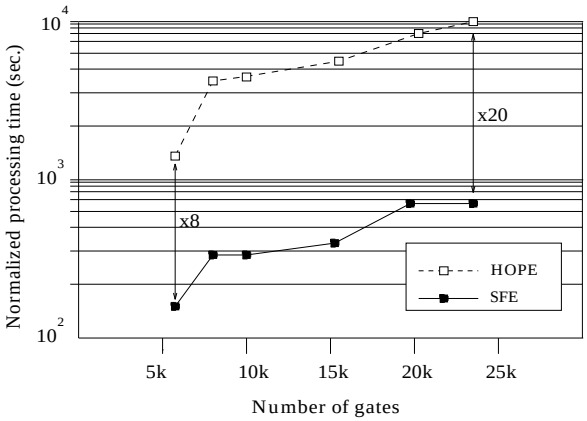


Fig. 6: Normalized Performance Comparison

Figure 6 shows the processing time for SFE and *HOPE*. The speedup of SFE over *HOPE* varies from 8 to 20. It is clear that SFE is especially advantageous for large circuits. For 100 K gate circuits, we can hope to reach a speedup of two orders of magnitude with respect to commercial tools. In contrast with HOPE, these tools are less efficient because they have to implement mechanisms to take into account user's libraries, memories or complex synchronization schemes.

## 6 CONCLUSIONS

An approach to evaluate test sets for large sequential circuits has been presented. This approach relies on the utilization of a hardware emulator to observe the effects of the faults on the circuits. Serial fault emulation takes advantage not only of the reconfigurability of Sram-based reprogrammable circuits but also of the reconfiguration speed of the *Meta Systems*'s hardware emulators.

The main advantage of serial fault emulation is that in contrast with software fault simulation the computing time is *quasi*-linear with the number of gates. So for large designs our approach can drastically reduce the time taken in the analysis of fault coverage, aliasing probability and detectability.

After *logic verification* and *fast prototyping*, test evaluation is a new application of hardware emulators that will encourage designer teams to adopt hardware emulator based methodology.

## 7 ACKNOWLEDGMENTS

The authors would like to thank Dong S. Ha of the University of Virginia for providing them with *HOPE* and B. Bailey of *Mentor Graphics* for its help in the preparation of this paper. They also thank E. Legai, J-S. Weil, F. Touzard and G. Morisset for their assistance with the *Meta Systems*'s CAP software.

## References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman "Digital Systems Testing and Testable Design", *New York, W.H. Freeman and Company, 1990*, p. 134

[2] P. S. Bottorff "Test Generation and Fault Simulation", *VLSI Testing, North Holland Ed., 1985*, pp. 29-64

[3] R.K. Brayton, G.D. Hatchel and A.L. Sangiovanni-Vincentelli "Multilevel Logic Synthesis", *Proc. of the IEEE, Vol. 78, No 2, Feb. 1990*, pp. 264-300

[4] M. Butts, J. Bacheler and J. Varghese "An Efficient Logic Emulation System", *Proc. ICCD, 1992*, pp. 138-141

[5] W. T. Cheng and M.L. Yu "Differential Fault Simulation - A Fast Method Using Minimal Memory", *Proc. 26th DAC, 1989*, pp. 424-428

[6] S. Gai and P. L. Montessoro "Creator : New Advanced Concepts in Concurrent Simulation", *IEEE Trans. on CAD, Vol 13, No 6, June 1994*, pp. 786-795

[7] J. Gateley *et al.* "UltraSPARC-I Emulation", *Proc. 32nd DAC, 1995*, pp. 535-540

[8] D. Harel and B. Krishnamurthy "Is There Hope for Linear Time Fault Simulation ?", *Fault Tolerant Computing Symposium, July 1987*, pp.28-33

[9] D.D. Hill and D.R. Cassiday "Preliminary Description of Tabula Rasa, an Electrically Reconfigurable Hardware Engine", *Proc. ICCD, Sept. 1990*, pp. 391-395

[10] H-C. Hsieh *et al.* "A Second Generation User-Programmable Gate Array", *Proc. Custom Integrated Circuit Conference, 1987*, pp. 515-521

[11] U. R. Khan, H.L. Owen and J. L. A. Hughes "FPGA Architectures for ASIC Hardware Emulator", *Proc. 6th IEEE ASIC Conference, 1993*, pp. 336-340

[12] H.K. Lee and D.S. Ha "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits", *Proc. 29th DAC, 1992* pp. 336-340

[13] H.K. Lee and D.S. Ha "New Techniques for Improving Parallel Fault Simulation in Synchronous Sequential Circuits", *Proc. ICCAD, 1993* pp. 10-17

[14] E. W. Thomson and S. A. Szygenda "Parallel Fault Simulation", *Computer, Vol. 8, No 3, March. 1975*, pp. 177-188

[15] E. G. Ulrich and T. Baker "Concurrent Simulation of nearly Identical Digital Networks", *Computer, Vol. 7, April 1974*, pp. 204-209

[16] N. Van Brunt "The Zycad Logic Evaluator and its Application to Modern System Design", *Proc. ICCD, 1983*, pp. 232-233

[17] S. Walters "Computer-aided Prototyping for ASIC-based Systems", *IEEE Design and Test, June 1991*, pp. 4-10