Effective emulation of permanent faults in ASICs through Dynamically Reconfigurable FPGAs

Ernesto Sanchez, Luca Sterpone, Anees Ullah
Dipartimento di Automatica e Informatica
Politecnico di Torino
Torino, Italy

Abstract— Hardware fault emulation for Application Specific Integrated Circuits (ASICs) on FPGAs can considerably reduce the time required for the fault simulation. This paper presents a methodology to emulate ASIC faults on state-of-the-art FPGAs. The fault emulation is achieved by following a fully automated process consisting of: constrained technology mapping of ASIC net-list; creation of fault dictionary, generation of faulty partial bit-streams and fault emulation. The proposed approach exploits run-time partial reconfiguration techniques for fault injection and avoids full net-list re-compilations. The method’s feasibility is assessed through carefully selected circuits and overhead in terms of area and timing is reported.

Keywords—Software Fault Simulation (SFS), Hardware Fault Emulation (HFE), Run-Time Reconfiguration (RTR)

I. INTRODUCTION

Photo-lithography driven VLSI chip fabrication technology is reaching its physical limits due to aggressive technology scaling towards nano-metric dimensions. These limitations manifest itself in form of manufacturing defects like stuck-at, stuck-open and bridge fault to name a few. For the fast moving Application Specific Integrated Circuits (ASIC) industry increasing the yield within stringent time-to-market requirements is essential. Fault simulation is a mandatory step to determine the yield of any VLSI chip fabrication process. However, the time required for fault simulation is prohibitively huge. One reason of lengthy simulation times is the adoption of software based mechanisms. Although they provide flexibility but are inherently slow due to sequential nature of computations. An alternative solution is to use hardware based fault emulation on state-of-the-art reconfigurable Field Programmable Gate Arrays (FPGAs) which can greatly reduce the timing requirements.

Two widely used techniques for fault emulation of VLSI circuits on FPGAs are circuit instrumentation and circuit reconfiguration. The circuit instrumentation approaches modifies the original net-list by inserting extra-hardware elements for fault injection purposes. In contrast, the reconfiguration-based techniques change the configuration bit-stream to inject faults and therefore do not incur hardware overhead. Although reconfiguration based approaches are very attractive, the challenge is how to fault emulate every ASIC fault with a corresponding faulty configuration bit-stream for the FPGA.

For fault emulation of ASICs on FPGA, it is necessary to consider the types of technology nodes both utilize. ASIC net-lists are composed of gates in a specific standard cell library while FPGA net-lists comprise of Look Up Tables (LUTs). LUTs can implement any combinational Boolean function of “k” inputs where “k” is the maximum number of inputs to the LUT. To optimize area (usually measured in number of LUTs) and delay, during mapping of the ASIC net-list on FPGA, the FPGA CAD tools changes the structure of the original ASIC net-list; clearly, this process is not affordable for fault emulation purposes, since it may lead to structural differences in the considered circuits, modifying the actual faults to be emulated. A possible solution may rely on partitioning the ASIC net-list to a set of circuit chunks suitable to be placed into the FPGA LUTs; however, this division process is hard to obtain by using commercial FPGA tools. Therefore, a known technology mapping from the ASIC net-list to FPGA net-list is mandatory for maintain the same circuit structure, that guarantees the actual emulation of all the ASIC faults.

The authors in [1] used a constrained technology mapping exploiting a commercial fault emulation platform for serial fault emulation. This technique results in a known logic mapping for each gate of the original ASIC gates. Resultantly, the fault emulation for each ASIC fault with a corresponding LUT configuration string is made possible. However, this technique was applied using a commercial fault emulation platform requiring vendor specific tool-chain.

This paper presents an efficient methodology for the stuck-at fault emulation of ASICs on commercial FPGAs. The methodology starts from an ASIC synthesized net-list and applies a constraint technology mapping to get the circuit without any modification into an FPGA net-list comprising of LUTs. Then, a fault dictionary generator automatically computes all the faulty LUT strings. Consequently, a partial bit-stream is generated for the every fault in the obtained dictionary, which is used later for fault emulation on FPGA. Our methodology enables the run-time injection of faults using dynamic partial reconfiguration and avoids lengthy recompilation times. Experimental results gathered on a set of representing circuits show that the approach is able to achieve a fault emulation speedup compared with software fault simulation while guaranteeing...
the emulation of every ASIC fault with a corresponding FPGA configuration string.

The rest of the paper is organized as following. Section II summarizes some of the most relevant works related to FPGA-based fault emulation. Section III motivates the needs of why we need to adopt a custom technology mapping. Section IV explains the LUT fault emulation model, whereas Section V discusses in details the developed approach and implementation flow. Section VI supports the proposed methodology with experimental results; and finally, Section VII concludes the paper.

II. RELATED WORK

In this section, we outline some of the most relevant techniques used for emulating ASIC faults on reconfigurable hardware based on FPGAs. These methods can be mainly classified according to the adopted fault injection methodology.

A. Circuit instrumentation based approaches

Techniques based on circuit instrumentation are intrusive in nature and modifies the structural descriptions of circuit by adding extra hardware for fault injection. A fault injection scan chain is used by the authors in [2, 3, 4] enabling them to avoid time-consuming re-compilation steps for generating fault bit-streams. However, the size of fault list is directly proportional to the hardware complexity of the injector circuitry, introducing a bottleneck for large VLSI circuits. A different technique that combines recompilation and circuit instrumentation is presented in [5]; a fault is injected in the ASIC net-list and then compiled to generate the faulty bit-stream. Independent faults are identified and instrumented in such a way to inject multiple faults at the same time reducing the reconfiguration time.

B. Reconfiguration based approaches

Reconfiguration based fault injectors modify the configuration bit-stream of the design to emulate faults. The authors in [1] utilize a commercial fault emulation platform that constrains the technology mapping of logic cones to LUTs and generates a corresponding bit-stream for each fault in the logic cone in form of FPGA reconfiguration list.

The authors show that for designs with more than 100,000 gates hardware emulation is two times fast compared to software fault simulation. JBits based tool-flow [6] is used for directly changing the configuration bits of a CLB in order to inject faults by authors in [7]. However, JBits is no longer supported for state-of-the-art commercial FPGAs. A direct bit-stream manipulation based injection is used by the authors in [8, 9] to inject faults in LUTs without constraining technology mapping. They reduce the size of fault list by considering only the active inputs of LUTs. However, this technique does not guarantee that every ASIC fault will be covered. The fault injection for a wide variety of fault models was presented in [10] where the authors develop faulty bit-stream by changing the HDL models. This requires time-consuming re-compilation for each injection of the fault model. Another interesting work exploiting partial reconfiguration for fault injection is presented in [11]. The authors present a methodology for the correlation of stuck-at fault model with that of Single Event Upset (SEU) fault model. The work presented in this paper takes an approach similar to [1] in quest to achieve a guaranteed emulation of ASIC permanent faults. However, we present a general method that can be applied to any FPGA device without resorting to very costly commercial fault emulation platforms and vendor tools.

III. FAULT EMULATION ON FPGAS

The emulation of ASIC faults on FPGA requires the knowledge of the FPGA’s resource that can be exploited for fault injection. This section outlines the generic architecture of FPGA devices that may be exploited during fault emulation. In particular, combinational logic fault injection based on LUT reconfiguration, as well as, flip-flop’s fault injection techniques are summarized. In addition, some considerations about the relationship between technology mapping and its effects on fault list are described in details.

A. Emulating Faults for LUTs and FFs

FPGA consists of tiles of different primitive types. For example, a tile can be made of Configurable Logic Blocks (CLBs), DSPs, BRAMs, IOBs and Interconnect Tile. The CLBs are the workhorse of FPGAs for implementing combinational and sequential elements of a circuit. Each CLB consists of a number of function generators in form of LUTs. Within each CLB, flip-flops reside in close proximity to LUTs. Each CLB can connect to global horizontal and vertical interconnect lines using the Interconnect Tile located near to it. For a LUT with “k” inputs the maximum number of configurations is $2^k$. Fig. 1 shows an abstract architectural view of a part of CLB and its Interconnect Tile. For illustration purposes only one LUT and FF is shown in the Fig. 1. It can be noted that the LUT implements a six input function with the following Boolean equation

$$O_6 = (((((00 \text{ and } 11) \text{ and } 15) \text{ and } (I_2 \text{ and } (I_3 \text{ or } I_4)))'))$$
If this equation represents a logic cone that belongs to the original ASIC net-list then, it is possible to emulate ASIC equivalent faults on the structure through a corresponding LUT equation modification. For example, for net “f” in Fig. 1, a stuck-at zero corresponds to the following LUT equation:

\[ O_6 = ((i_0 \text{ and } i_1)' \text{ and } i_5)' \text{ and } (1)' \]

Flip-flop faults emulation, on the other hand, may be handled resorting to multiple scenarios according to the circuit structure. Fig. 1 shows a FF that can be fed by the “O6” output of a LUT, “O5” output of a LUT or with the auxiliary input line “AX”. For emulating a fault at the input of the FF, a LUT or an Interconnect Tile can be used. When the FF is connected to a single LUT within the same CLB, the LUT equation can be modified accordingly to the considered fault model to emulate the fault at the input of the FF. However, when the LUT output feeds multiple sinks including a FF, this strategy is not suitable. In this case, it is possible to use the “GROUND” and “VCC” resources present near the Interconnect Tile of each CLB, by connecting the “AX” to the input of the FF using the Interconnect Tile and the configurable multiplexer available within each CLB.

B. Technology Mapping and Fault Emulation

FPGA CAD tools significantly differ from ASIC tools because the underlying technology both utilize is very different. Emulating ASIC faults using a FPGA based method requires maintaining the structure of the ASIC net-list. However, resorting to standard FPGA tools may result in different circuit structure since these tools usually make several logic optimizations modifying the ASIC net-list. The mapping of ASIC gates to LUTs, is usually referred to as K-LUT technology mapping. This process may duplicate part of the original ASIC gates for reducing the number of LUTs or reducing delay or both [12]. Another optimization uses functional decomposition of ASIC gates to smaller gates for reducing the number of LUTs [13]. However, it is not possible to know how the FPGA tools partitioned the ASIC net-list to mapping to LUTs. In any case, the different approaches for K-LUT technology mapping combine multiple ASIC gates into one FPGA LUT, while maintaining the circuit flip-flops. Thus, it is possible to exploit the similarity between the two net-lists for fault emulation purposes. However, for guaranteed fault emulation (i.e., every ASIC fault has a corresponding LUT configuration) the behavior of both the ASIC circuit and the FPGA model must be the same for all possible inputs. Using this criteria for finding the equivalent LUT configuration for each ASIC fault using commercial software CAD tools for FPGA is hugely time consuming. Consider the case when multiple LUTs reside in a combinational logic between two flip-flops. For finding the equivalent LUT configuration for an ASIC fault the size of search space is the following.

\[ N_{\text{PIS}} \times (N_{\text{fan}} \times 2^k) \]

Where “N_{\text{fan}}” represents the number of LUTs that resides in the combination logic between two flip-flops, “N_{\text{PIS}}” is the number of primary inputs and \(2^k\) is the number of possible configuration for each LUT. Obviously, for guaranteeing fault emulation this is prohibitively time consuming task, and diminishes the returns of hardware emulation when all faults are considered. The search space for this problem can be limited by using the active number of inputs to the LUT as presented in [8, 9]. However, for guaranteed fault emulation it is still excessively large. Therefore, it is necessary to use a custom technology mapping that avoids changes to net-list and results in a known mapping of ASIC gates to FPGA LUTs.

IV. PROPOSED METHODOLOGY

In this section, we discuss in detail the proposed methodology for ASIC fault emulation on state of the art Xilinx FPGAs. The methodology consists of two main phases: a custom technology mapping of the ASIC net-list to LUT-level FPGA net-list, and the creation of a fault dictionary as shown in Fig. 5. The developed tools use Boost C++ libraries, and builds upon the framework presented in [14]. The following paragraphs explain in details all the phases involved.

A. Custom Technology Mapping

The goal of this step is to translate an ASIC gate-level net-list to a LUT-level FPGA net-list suitable for fault emulation. This problem of converting the ASIC gates to LUTs is usually called K-LUT technology mapping. The existence of multiple fan-out nodes makes the problem of optimal technology mapping very challenging [15].

Fig. 2 shows an ASIC net-list with a couple of multiple fan-out gates. The circuit is then mapped to three FPGA LUTs, represented in Fig. 2 by the boxes labeled as LUT1, LUT2, and LUT3. This technology mapping uses the concept of Maximum Fan-out Free Cones (MFFCs) [16]. The MFFC of a node “v” represents the maximum number of nodes in the transitive fan-in of a gate in such a way that the fan-out of every node in the MFFC except the node “v” is inside the MFFC. For example, the MFFC of node “k” represented by MFFC_3 is composed of all the gates in the transitive fan-in
“h,i,b,c” except the node “a” because it fan-outs to \textit{MFFC}\textsubscript{j}. It is interesting to note that the gate “b” has also multiple fan-out but they re-converge on gate “k” and therefore are a part of \textit{MFFC}\textsubscript{k}. If the \textit{MFFC} is k-feasible (i.e., the number of inputs are less than or equal to k, the maximum number of inputs to a LUT) it can be collapsed into a LUT. All the \textit{MFFCs} in Fig. 3 are k-feasible and therefore are a candidate for a LUT. The advantage of \textit{MFFC} based mapping is that emulating a fault on a multiple fan-out node requires reconfiguring only a single LUT and thus can exploits collapsed ASIC fault list to reduce the fault emulation time. However, the required number of LUTs can be reduced from three to two if duplication is allowed as shown in Fig. 3. It can be noted that node “a” has been duplicated. For fault emulation purposes every ASIC fault on the structure of node “a” should be emulated in both LUTs. With state of the art ASIC synthesis tools it is often the case that a gate fan-outs to more than two gates. Resultantly, allowing duplication during technology mapping would mean that a single ASIC fault is converted to multiple FPGA faults.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig3.png}
\caption{Mapping with duplication.}
\end{figure}

In the proposed approach, the presented mapping process uses \textit{MFFC} duplication-free mapping of the ASIC net-list avoiding nodes duplication in the resulting LUTs. In order to create the actual LUT-level net-list, it is required to identify the initial values every LUT is going to assume. Let us describe how this process is performed for the LUT 2 in Fig.3. The input nets “in1”, “in2”, “in3” and “fan_out_wire” are arbitrarily bound to the LUT address lines “I0”, “I1”, “I2” and “I4”. For a 6-input LUT, the address lines can be represented as 64-bit vectors formed by concatenating the bits along the truth table from top to bottom. These address lines can be used to calculate the LUT function (often called \textit{LUT INIT value}) by walking the LUT’s DAG and applying the corresponding Boolean operations in the following manner.

\begin{align*}
I0 &= \text{AAAA}_\text{AAAA}_\text{AAAA}_\text{AAAA} \\
I1 &= \text{CCCC}_\text{CCCC}_\text{CCCC}_\text{CCCC} \\
\text{(nand)} &= \text{7777}_\text{7777}_\text{7777}_\text{7777} = \text{b\_out} \\
I2 &= \text{FOFO}_\text{FOFO}_\text{FOFO}_\text{FOFO} \\
I4 &= \text{FFFF}_\text{0000}_\text{FFFF}_\text{0000} \\
\text{(nand)} &= \text{0F0F}_\text{FFFF}_\text{0F0F}_\text{FFFF} = \text{c\_out}
\end{align*}

Now for calculating the values of next level of gates, we can utilize the above computed values:

\begin{align*}
\text{b\_out} &= \text{7777}_\text{7777}_\text{7777}_\text{7777} \\
I0 &= \text{AAAA}_\text{AAAA}_\text{AAAA}_\text{AAAA} \\
\text{(nand)} &= \text{DDDD}_\text{DDDD}_\text{DDDD}_\text{DDDD} = \text{h\_out}
\end{align*}

For gate “i” we can proceed as

\begin{align*}
\text{c\_out} &= \text{0F0F}_\text{FFFF}_\text{0F0F}_\text{FFFF} \\
\text{b\_out} &= \text{7777}_\text{7777}_\text{7777}_\text{7777} \\
\text{(or)} &= \text{7F7F}_\text{FFFF}_\text{7F7F}_\text{FFFF} = \text{i\_out}
\end{align*}

The LUT \textit{INIT} value is the following

\begin{align*}
\text{i\_out} &= \text{7F7F}_\text{FFFF}_\text{7F7F}_\text{FFFF} \\
\text{h\_out} &= \text{DDDD}_\text{DDDD}_\text{DDDD}_\text{DDDD} \\
\text{(nand)} &= \text{A2A2}_\text{2222}_\text{A2A2}_\text{2222} = \text{INIT}
\end{align*}

Using this approach a LUT initialization value is calculated and the resultant description is translated to the required format to be mapped into the FPGA. The standard FPGA CAD steps are performed for the implementation of the golden version, followed by bitstream generation phase.

\textbf{B. Fault Dictionary Creation}

The purpose of the fault dictionary creation phase is to generate partial bitstreams for all the faults in the collapsed ASIC fault list. The ASIC fault list may be derived using a commercial fault simulator. The proposed algorithm, called Fault Dictionary Algorithm (FDA) requires a post placed and routed FPGA simulation model, as well as the ASIC logic cones data-base generated during custom technology mapping and the collapsed ASIC fault list. For every ASIC logic cone represented as a Directed Acyclic Graph (DAG), the FDA creates a corresponding faulty LUT equation as shown in Algorithm 1. The algorithm utilizes a map data-structure called “VMap” which is indexed by a net and the value stored is the current Boolean equation for that net during the topological graph traversal. The Boolean equation is generated by traversing the graph in forward topological order from primary inputs to primary outputs and at the output of each logic gate, the type of operation represented by the gate is applied to the values indexed by the input nets. The resulting value is stored in the map data-structure indexed by the output net. For nets which are currently at a stuck-at for fault emulation purposes the computed values during graph traversal is ignored and the stuck-at fault equivalent value is stored in the map data-structure. The resulting equation is generated in a format compatible with Xilinx Design Language (XDL). It is important to retain LUT address lines even in cases where a fault may make the resultant equation independent on some particular inputs. For example, in Fig. 1, a stuck-at one fault on “g” net makes the whole LUT equation independent on “I0” and “I1” address lines; however, it is important to keep these signals in the equation so that FPGA routing is kept
Algorithm 1: Fault Dictionary Creation (FDA) algorithm

Input: ASIC Logic Cones and ASIC Collapsed Fault List
Output: Faulty Bitstream Dictionary

1. foreach NET € LUT DAG do
2.   foreach faultCondition on NET do
3.     if(faultCondition on NET € ASIC faultlist) do
4.       foreach NET € LUT DAG do
5.         VMap[NET]=FaultyHexValue;
6.     end_foreach
7.   end_foreach
8. end_foreach

Fig 4. The flow of fault dictionary creation algorithm.

V. EXPERIMENTAL SETUP AND RESULTS

The experimental results were collected on a Xilinx Virtex-5 5VLX110T FPGA emulating the stuck-at faults for every ASIC benchmark circuit. The methodology is applied to 5 circuits including 3 arithmetic circuits and 2 DSP circuits. A Micro-blaze based version of the platform presented in [17] was adopted for test pattern application to the Design Under Test (DUT). The DUT has the custom technology mapped ASIC net-lists in two version i.e. the golden copy and the faulty copy. The DUT also contains a hardware comparator that compares the output from both copies to detect a faulty condition. The DUT is attached as an IP core to the Micro-blaze processor for test pattern application and monitoring. The FDA algorithm’s generated faulty LUT equations are inserted into XDL file of the whole Micro-blaze system to generate valid partial bit-streams. The stored partial bit-streams are downloaded one by one from a personal computer to the FPGA. The processor then starts to apply the test patterns that were derived from the commercial fault simulator stored in an off-chip memory. The results of fault detection are communicated back to the host PC which proceeds with the injection of another partial faulty bit-stream. Another set of experiment utilizing the post-Placed and Routed Simulation model used ModelSim for fault simulation of the ASIC net-list, Xilinx mapped net-list and custom mapped net-list to show the relative difference in speeds. It is important to note that the test patterns generated with commercial fault simulator stored in STIL format are used with ModelSim. The fault injection on the ASIC net-list is achieved with ModelSim “force freeze” command while for injecting LUT faults in FPGA net-list, re-compilation with ModelSim “-G” switch is performed. Therefore, the simulation timing required for FPGA is more than that of ASIC fault injection due to re-compilation requirements. The fault simulation time “T_fpga” is calculated by using ModelSim “simstats” command with an automated TCL script. The actual timing requirements of our custom mapper using hardware emulation are given by “T_fpga” which is calculated in the following way.

\[ T_{fpga} = T_{comp} + T_{config} + N_{cf} (T_{reconfig} + T_{pd} N_{p}) + T_{reconfig} N_{Luts} \]

Let “T_comp” be the time required for compilation of the golden version of the circuit, “T_config” be the bit-stream download time, “N_cf” be the number of collapsed faults required for ASIC fault simulation, “T_reconfig” be the fault injection or fault removal time, “T_prop” is the propagation delay of the critical path in case of combination circuit or the clock period length in case of sequential circuit and “N_p” are the average number of patterns required for detection of faults in a benchmark circuit. Then we can write the total fault emulation time considering serial fault emulation as follows.

It can be noted that the compilation time and configuration time are required only once and therefore they can be neglected. The reconfiguration time depends on the number of LUTs because for emulating faults in a single LUT there is no need to remove the fault before injecting another one but if the next fault is to be injected in another LUT then the previous fault LUT should be restored to the golden LUT.
configuration value. The product of $T_{pd}^*N_p$ is the time required for fault emulation of each fault. Table II presents the fault statistics from the original ASIC net-list using commercial fault simulator Tetra-Max. The original ASIC design was synthesized with Design Vision for pdt2002 library. It should be noted that because of the known technology mapping we are able to directly use the values from Table II and we achieve the same coverage for the stuck-at fault model given by the Tetra-Max on the given benchmark circuits. Although our custom technology mapping based fault emulation flow enables guaranteed emulation of every ASIC fault in the ASIC fault list with a reduction in fault simulation timing requirements, however, this is achieved at an overhead in terms of area and delay of the circuit as reported in Table III. It can be noted that due to MFMC based mapping our mapper utilizes more LUTs and runs more slowly than the Xilinx mapper.

<p>| Table I. Fault Emulation/Simulation Times |</p>
<table>
<thead>
<tr>
<th>Circuit</th>
<th>ASIC $T_{model}(nsecs)$</th>
<th>FPGA $T_{model}(nsecs)$</th>
<th>Custom Mapper $T_{model}(nsecs)$</th>
<th>Custom Mapper $N_c$</th>
<th>LUTs</th>
<th>FFs</th>
<th>Delay (nsecs)</th>
<th>Custom Mapper $F_H$</th>
<th>LUTs</th>
<th>FFs</th>
<th>Delay (nsecs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adder64</td>
<td>314</td>
<td>320.36</td>
<td>32.58</td>
<td>0.004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplier32</td>
<td>358.76</td>
<td>10,124.09</td>
<td>351.97</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplier64</td>
<td>618.71</td>
<td>1,106,063.21</td>
<td>1,148.99</td>
<td>3.73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIB filter</td>
<td>120.12</td>
<td>24,702.26</td>
<td>418.17</td>
<td>0.058</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hilbert transform</td>
<td>28.15</td>
<td>796.11</td>
<td>92.42</td>
<td>0.012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| Table II. Fault Statistics for ASIC net-list |</p>
<table>
<thead>
<tr>
<th>Circuit</th>
<th>$N_c$</th>
<th>$N_p$</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adder64</td>
<td>2,436</td>
<td>1,410</td>
<td>100.00</td>
</tr>
<tr>
<td>Multiplier32</td>
<td>42,304</td>
<td>23,431</td>
<td>137.68</td>
</tr>
<tr>
<td>Multiplier64</td>
<td>170,612</td>
<td>94,125</td>
<td>288.00</td>
</tr>
<tr>
<td>FIB filter</td>
<td>46,696</td>
<td>27,873</td>
<td>14.00</td>
</tr>
<tr>
<td>Hilbert transform</td>
<td>11,168</td>
<td>6,179</td>
<td>45.00</td>
</tr>
</tbody>
</table>

<p>| Table III. Area and delay overhead |</p>
<table>
<thead>
<tr>
<th>Circuit</th>
<th>Xilinx Mapper</th>
<th>Custom Mapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>FFs</td>
<td>Delay (nsecs)</td>
</tr>
<tr>
<td>Adder64</td>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td>Multiplier32</td>
<td>2,624</td>
<td>0</td>
</tr>
<tr>
<td>Multiplier64</td>
<td>7,110</td>
<td>0</td>
</tr>
<tr>
<td>FIB filter</td>
<td>2,052</td>
<td>971</td>
</tr>
<tr>
<td>Hilbert transform</td>
<td>288</td>
<td>225</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS AND FUTURE RESEARCH

This paper presented a methodology to emulate ASIC permanent faults using FPGA. A novel fault emulation flow is presented. The flow utilizes a custom technology mapping for directly converting the post layout gate-level net-list into a LUT level net-list. This known mapping of gates to LUTs enables us to develop a fault dictionary from the ASIC fault list to FPGA partial bitstreams. The methodology avoids drastic changes to the net-list, therefore, does not need lengthy re-compilation times during thefault emulation process. Furthermore, our experimental results demonstrate a significant speed up for fault emulation compared to software based fault simulation. In future, we would like to extend the methodology to automatic test pattern generation and fault behavior analysis for a wide set of permanent and dynamic fault models.

REFERENCES


