# Reversing FPGA architectures for speeding up fault injection: does it pay?

Ilya Tuzov, David de Andrés and Juan-Carlos Ruiz
ITACA, Universitat Politècnica de València, Campus de Vera s/n, 46022, Spain
Email: {tuil, ddandres, jcruizg}@disca.upv.es

***Abstract*—Although initially considered for fast system prototyping, Field Programmable Gate Arrays (FPGAs) are gaining interest for implementing final products thanks to their inherent reconfiguration capabilities. As they are susceptible to soft errors in their configuration memory, the dependability of FPGA-based designs must be accurately evaluated to be used in critical systems. In recent years, research has focused on speeding up fault injection in FPGA-based systems by parallelising experimentation, reducing the injection time, and decreasing the number of experiments. Going a step further requires delving into the FPGA architecture, i.e. precisely determining which components are implementing the considered design (mapping) and which are exercised by the considered workload (profiling). After that, fault injection campaigns can focus on those components actually used to identify critical ones, i.e. those leading the target system to fail. Some manufacturers, like Xilinx, identify those bits in the FPGA configuration memory that may change the implemented design when affected by a soft error. However, their correspondence to particular components of the FPGA fabric and their relationship with the implementation-level model are yet unknown. This paper addresses whether the effort of reversing an FPGA architecture to filter out redundant and unused essential bits pays in terms of experimental time. Since the work of reversing the complete architecture of an FPGA is titanic, as the first step towards this ambitious goal, this paper focuses on those elements in charge of implementing the combinational logic of the design (Look-Up Tables). The experimental results that support this study derive from implementing three soft-core processors on a Zynq SoC FPGA and show the interest of the proposal.***

***Keywords*—FPGA, essential bits, fault injection**

## I. Introduction

Field Programmable Gate Arrays (FPGAs) have become increasingly widespread across diverse application domains, ranging from high-performance computing clusters to aerospace equipment. Such valuable features of FPGAs explain this trend as energy efficiency, short design cycle, and run-time reconfiguration capabilities. On the downside, the usage of FPGAs leads to an increased number of dependability and security threats, as FPGAs are highly susceptible to Single Event Upsets (SEU) affecting their Configuration Memory (from now on CM) [1]. As each cell of the CM directly controls the configuration or interconnection of fabric elements to provide the functionality required by a design, any modification of its content directly impacts the implemented design. This is why SEUs affecting the FPGA CM (typically SRAM) manifest as stuck-at faults from the viewpoint of the implemented design, as they may lead to open lines, shortcircuits, delays, and permanent changes in the combinational logic of the circuit. Once affected by SEUs, CM cells remain faulty until partially or fully reprogrammed to fix the problem, using scrubbing techniques [2], or to implement another circuit. This problem becomes critical in high-radiation environments [3] [4], where upset rates may exceed the terrestrial conditions by orders of magnitude. Thus, designers should carefully assess the robustness of FPGA-based systems against SEUs to locate their weak points, which should be protected in the first place, and evaluate the efficiency of any integrated SEU mitigation mechanism [5].

Grant PID2020-120271RB-I00 funded by MCIN/AEI/ 10.13039/501100011033.

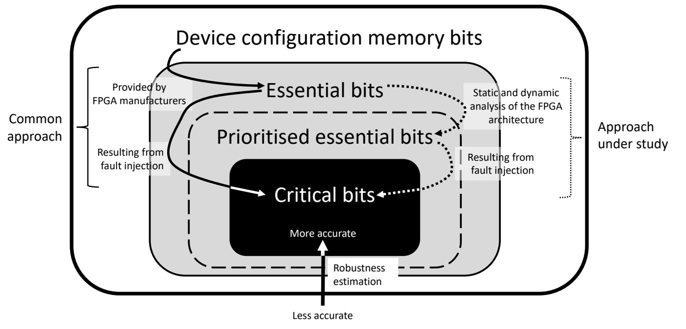

Device configuration memory bits
Common approach
Provided by FPGA manufacturers
Resulting from fault injection
Essential bits
Prioritised essential bits
Critical bits
More accurate
Robustness estimation
Less accurate
Static and dynamic analysis of the FPGA architecture
Resulting from fault injection
Approach under study

Fig. 1. Relationship of device configuration memory bits and design bits. Solid lines denote the common approach, whereas dashed lines denote the approach under study.

The robustness of FPGA-based designs is commonly estimated after the reliability device report provided by manufacturers. It is calculated by scaling the failures-in-time (FIT, failures in 1 billion device-hours of operation) per megabit of the device by the number of megabits existing in the FPGA CM. However, designs loaded into FPGAs typically only use a fraction (around 10-35%) of the total number of available megabits (see case studies for convolutional neural networks in [6]). Even though designs may use most of the available logic resources, routing resources account for about 70% of the CM bit and they will not be used in the same proportion. Those unused bits are considered non-essential for the design and could be disregarded during robustness estimation. Some works, like [7], obtain rough estimates by scaling the total CM bits of the selected device by the configurable logic blocks utilisation ratio of the design. Although fast for calculation, these methods are highly inaccurate and may result in insufficient

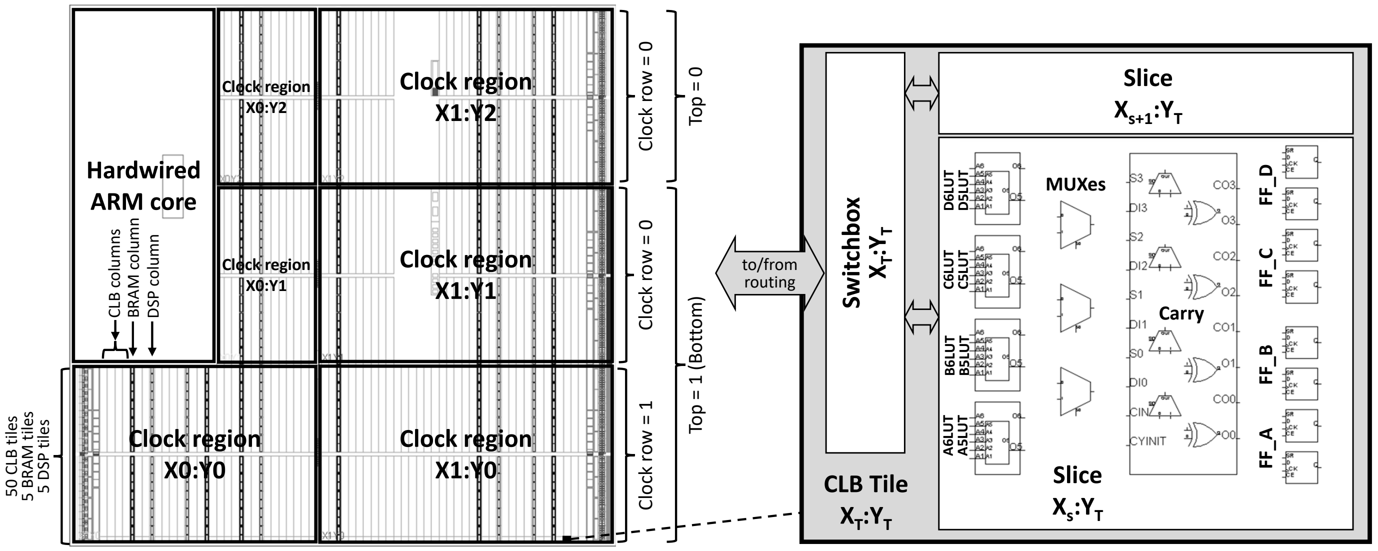


Fig. 2. Architecture of Xilinx 7-Series and Zynq-7000 FPGAs. The distribution of clock regions, clock rows and top/bottom of a Zynq xc7z020clg484-1 device is shown as an example.

protection against SEUs or unnecessary overprotection at the cost of performance, power, and area overheads.

Xilinx *essential bits* technology identifies those CM bits that are involved in the circuitry of the design implemented in an FPGA (in grey in Fig. 1). More elaborated proposals [8] [9] make use of this technology to restrict the robustness estimation to just those essential bits. Nevertheless, not all upsets on these essential bits will affect the functionality provided by the design, and thus, part of them may be disregarded to accelerate experimentation. According to [10] the effective soft error rate is 5% (1 in 20 upsets causes a functional soft error) on average and never greater than 10% in the worst case.

Determining the complete list of *critical bits*, i.e. those bits leading the system to fail, requires an exhaustive fault injection campaign targetting every essential bit. Although several works, like [11], use fault injection to identify the set of critical bits in a design and obtain highly accurate robustness estimations, there are still some improvements that could be worthwhile pursuing. On the one hand, the essential bits provided by Xilinx technology include all CM bits from related logic primitives, even if they are only partially used by the design. This unnecessarily increases the number of fault injection experiments required to determine the set of critical bits. On the other hand, existing tools do not relate the set of essential bits with the design hierarchy.

Addressing the aforementioned challenges has the potential to reduce the set of essential bits to a smaller set of *prioritised essential bits* (see Fig. 1), thus, decreasing the number of fault injection experiments to carry out and the time devoted to their execution to determine the set of critical bits. In addition, this can be done without reducing the representativity of results since disregarded CM bits will be those not used by the design and not exercised by the considered workload. However, addressing this problem is very complex from a technical perspective, as the relationship between the FPGA fabric and its CM is usually unknown. The computational cost of reversing such a relationship may be greater than the time saved by reducing the set of essential bits initially under consideration.

This work studies the internals of FPGAs to determine where the bits corresponding to Look-Up Tables (LUTs) are located within the CM (mapping). The LUTs implement the combinational logic of designs and have been selected as a first step towards the challenging goal of mapping the complete set of CM bits of an FPGA. After a static (mapping) and dynamic (profiling) analysis, unused bits can be removed from the set of essential bits to reduce the number of fault injections required to estimate the robustness of the design.

First, Section II details the generic architecture of FPGAs, particularly that of the Xilinx 7-Series and Zynq-7000 devices that will be used as a target technology for this paper. After that, Section III-A proposes an algorithm that accurately locates LUTs bits within the CM of FPGAs and enables removing redundant essential bits from experimentation. Section III-B presents an approach for profiling the activity of LUTs to identify and remove inactive essential bits and prioritise the criticality of the rest of the CM bits according to their activity at runtime. The usefulness of the proposal is illustrated in Section IV through a case study of three soft-core processors implemented on a Xilinx Zynq-7000 SoC FPGA. Finally, conclusions drawn from this work and future research directions are summarised in Section V.

## II. Xilinx 7-Series and Zynq-7000 devices

A coarse-grain architecture of the programmable fabric of 7-series and Zynq-700 devices from Xilinx is depicted in Fig. 2. This section details this architecture following a bottom-up viewpoint.

TABLE I. Description of the frame address [12].

| Address Type | Bit Index | Description |
|---|---|---|
| Block Type | [25:23] | CLB, I/O, CLK (*000*), BRAM (*001*), CFG_CLB (*010*) |
| Top/Bottom Bit | 22 | Top rows (*0*) and Bottom rows (*1*) |
| Row Address | [21:17] | Clock row. It increases from centre to top, then resets and increases from centre to bottom. |
| Column Address | [16:7] | Major column, like a column of CLBs. Starts at 0 and increases from left to right (matches $X_T$). |
| Minor Address | [6:0] | Selects a frame within a major column. |

The logic elements inferred by synthesis tools are mapped to technology-specific components, known as *basic logic elements* (BELs) by Xilinx. For instance, look-up tables (LUTs) and flip-flops (FFs) implement the combinational and sequential logic of the design, respectively, whereas memory (BRAMs) and digital signal processing (DSPs) blocks improve the implementation of memories and multipliers. A set of related BELs defines a *Slice* that, for the considered families of devices, include four LUTs (A, B, C, and D), eight FFs, some multiplexers (MUXes), and a carry chain (CARRY). Slices appear in two flavours: *L* (logic), whose LUTs may implement combinational logic, and *M* (memory), whose LUTs may also act as distributed memory and shift registers.

A *configurable-logic block* (CLB) *Tile* comprises a pair of Slices and a *Switchbox*, which enables the connection of the Slices with the general routing logic. Tiles of the same type are arranged in columns that span the full height of the device, with BRAM and DSP columns interspersed with CLB columns.

A two-dimensional array of Tiles forms a *clock region*, which is horizontally divided into two parts by a clock lane. For the considered families of devices, clock regions consist of 50 CLB Tiles, 5 DSP Tiles, or 5 BRAM Tiles. Horizontally adjacent clock regions define a *clock row*.

Finally, different clock rows are joined together into the *Top* or *Bottom* part of the device.

Slices, Tiles, and clock regions are located in unrelated two-dimensional grids with $(X, Y)$ coordinates, thus providing a reasonably regular structure for the logic elements within the configurable fabric.

The contents of the CM of an FPGA determine the interconnection and configuration of fabric elements to provide the functionality required by the design. The CM is arranged in *frames* that are tiled about the device. These frames are the smallest addressable segments of the CM space, and each frame consists of 3232 bits (101 32-bits words) for the considered families of devices.

All read/write operations on the CM must act upon whole configuration frames, whose particular address must be written to the *Frame Address Register* (FAR) using any of the interfaces provided by the manufacturer. The five fields comprising the frame address are listed in Table I.

Even though part of the frame address is determined by the location of a logic element within the structure of the FPGA, the minor address related to that element is unknown. Likewise, the particular contribution of each bit of the frame to the configuration of the target element is also unknown. Thus, even though the Xilinx essential bits technology provides the location of all those bits identified as essential, no information is provided to determine to which of the implemented logic they refer or their purpose in the design.

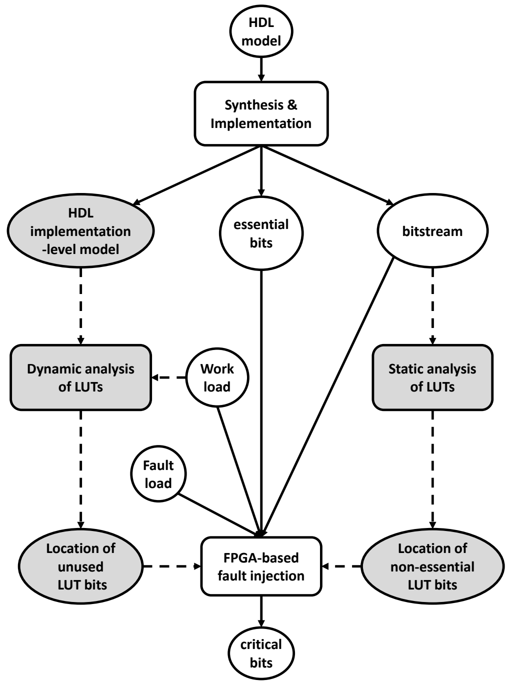

HDL model
Synthesis & Implementation
HDL implementation -level model
essential bits
bitstream
Dynamic analysis of LUTs
Work load
Static analysis of LUTs
Fault load
Location of unused LUT bits
FPGA-based fault injection
Location of non-essential LUT bits
critical bits

Fig. 3. Block diagram of the proposed approach (in grey) to optimise FPGA-based fault injection procedures for LUTs.

## III. Locating LUTs non-essential and unused bits

Approximately 15% of the total number of CM bits correspond to LUTS contents. Thus, locating the essential bits related to LUTs which are actually used by a design under a given workload may contribute to reducing the number of fault injection experiments required to determine the critical bits of a design. The procedures proposed in this approach, highlighted in grey in Fig. 3, make use of data extracted after implementing a design for the selected device.

### A. Static analysis of LUTs

*1) Are there redundant bits in LUTs?:* After synthesis, as shown in Fig. 4, the combinational logic of a design is represented by a set of logic primitives known as *LUT macrocells*. They are named $LUT\{6, \cdots, 2\}$, depending on the number of inputs of the implemented Boolean function $(I_5, \cdots, I_0)$, and $LUT6\_2$ for implementing two functions. They all have an *INIT* attribute (from 2 to 64 bits) representing the function's truth table.

During the implementation, in the selected families of devices, these LUT macrocells are mapped to 6-input LUTs (MUXes are used to implement Boolean functions up to eight variables). As depicted in Fig. 2, each of the four LUT BELs in a slice ($A6/A5$, $B6/B5$, $C6/C5$, $D6/D5$) presents

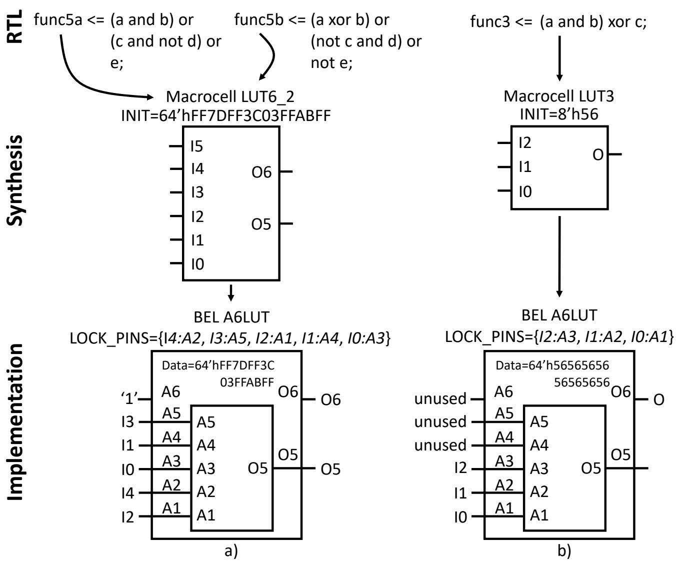
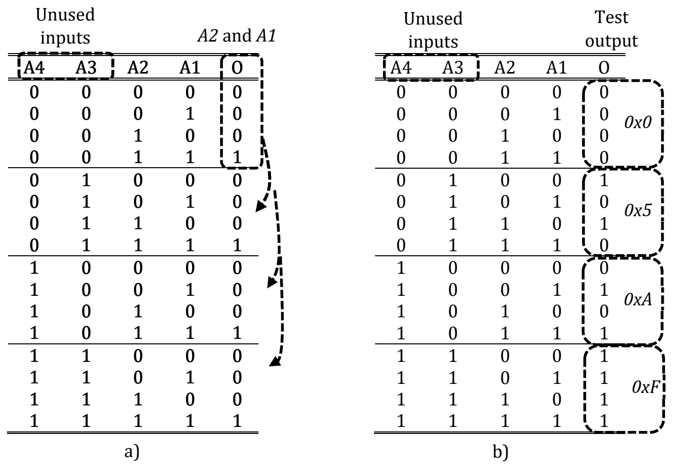

RTL
func5a <= (a and b) or (c and not d) or e;
func5b <= (a xor b) or (not c and d) or not e;
func3 <= (a and b) xor c;
Synthesis
Macrocell LUT6_2
INIT=64'hFF7DFF3C03FFABFF
I5
I4
I3
I2
I1
I0
O6
O5
Macrocell LUT3
INIT=8'h56
I2
I1
I0
O
Implementation
BEL A6LUT
LOCK_PINS={I4:A2, I3:A5, I2:A1, I1:A4, I0:A3}
Data=64'hFF7DFF3C 03FFABFF
'1'
I3
I1
I0
I4
I2
A6
A5
A4
A3
A2
A1
O6
O5
BEL A6LUT
LOCK_PINS={I2:A3, I1:A2, I0:A1}
Data=64'h56565656 56565656
unused
unused
unused
I2
I1
I0
O
a)
b)

Fig. 4. Example of how logic functions are synthesized as macrocells and then implemented as LUTs: a) two combined 5-input functions, and b) a 3-input function.


Unused inputs
A2 and A1
Unused inputs
Test output
0x0
0x5
0xA
0xF
a)
b)

a)

| A4 | A3 | A2 | A1 | O |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

b)

| A4 | A3 | A2 | A1 | O |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Fig. 5. a) Example of the contents of a 4-input LUT implementing $A2$ *and* $A1$, and b) test pattern used to determine the actual value of unused inputs.

six independents inputs ($A1, \cdots, A6$) and two independent outputs ($O5$ and $O6$). They can implement i) any six-input Boolean function, ii) two five-input Boolean functions that share common inputs, or iii) two Boolean functions with no more than 3 and 2 inputs [13]. To do so, it is necessary to adequately program the 64 bits of the CM representing the truth table of the desired Boolean functions. In the case of implementing two Boolean functions, $A6$ is set to logic '1', and the top 32 bits determine the $O6$ output, whereas the 32 bottom bits represent the $O5$ output (see Fig. 4a). Additionally, macrocell inputs can be assigned to any BEL inputs (*LOCK_PINS* attribute) to reduce the critical path.

When a LUT is used in a design, all 64 bits are highlighted as essential by the manufacturer as any change alters the implemented Boolean function. However, what if a function does not use all the available bits?

Let us consider, for instance, a $LUT2$ macrocell (*INIT* = $\{I1$ *and* $I0\}$, *LOCK_PINS* = $\{I1 : A2,\ I0 : A1\}$) implemented as a 4-input LUT BEL. Only 4 out of 16 bits of the LUT are required to implement the 2-input logic function. Nevertheless, as all CM bits must be programmed, the output of the function is replicated to completely fill the LUT, as shown in Fig. 5 (a). In such a way, the LUT will safely provide the expected result regardless of the value of the unused inputs $A4$ and $A3$. But, are all those bits really essential?

A simple design featuring one LUT was implemented on a real device to experimentally answer this question. After implementation, the contents of the LUT were modified using the editor provided by the manufacturer in such a way that, by going through all valid input values, a different pattern will be obtained as output depending on the actual value of the unused inputs. For instance, as can be seen in Fig. 5 (b), by sequentially providing as input 00, 01, 10, and 11, the obtained output should be 0x0, 0x5, 0xA, or 0xF depending on whether the unused inputs are held at 00, 01, 10, or 11 by the device. After testing LUTs with inputs ranging from 1 to 6 and located in several positions across the device, we experimentally concluded that unused bits are always physically bound to a logical '1' in the implementation.

Accordingly, only those LUT bits related to a high logic level of the unused inputs can be considered essential. The rest should not be considered for fault injection as they will never lead the system to fail. So, this means reducing the number of essential bits to 50%, 25%, 12.5%, 6.45%, or 3.125%, for 5-, 4-, 3-, 2-, and 1-input Boolean functions, respectively.

Although this could be used to improve the worst-case estimation of the design's failure rate, it is still necessary to establish the location of those bits within the CM to determine their criticality using fault injection.

*2) Locating LUTs bits in the CM:* Several works [14] [15] [16] have dealt with the extraction and replacement of LUTs content from the CM of FPGAs. However, none of them has proposed a generic algorithm to map each bit of a LUT's *INIT* attribute to the corresponding CM bit taking into account all LUT placement attributes (see Fig. 2). This is a must to enable filtering out the redundant bits from the LUT.

As before, a simple design featuring a single LUT is enough to obtain the required information experimentally. To ease the location of the whereabouts of the LUT contents in the CM, a zero-cold pattern (all bits are set to '1', except one of them that is set to '0') has been used for the *INIT* attribute of the $LUT6$ macrocell, whereas its *LOCK_PINS* attribute was set to $\{I5 : A6, I4 : A5, I3 : A4, I2 : A3, 1 : A2, I0 : A1\}$. This macrocell was mapped to an $A6LUT$ BEL located at Slice ($X_S$, $Y_T$), CLB Tile ($X_T$, $Y_T$), clock row ($X_{CR}$, $Y_{CR}$), and Top ($T$). A total of 64 variations of this design were implemented, shifting the '0' from the bit 0 to 63 of the *INIT* attribute to determine the exact location of each bit of the LUT

TABLE II. Excerpt of the LUT6 macrocell mapping within the CM.

| INIT (64 bits) | CM Fragment (64 bits): Top = 1, Clock row = 1, Column = 18, Word = 36, Bits [15:0] | | | |
|---|---|---|---|---|
| LUT_A BEL ($X_{113}$, $Y_{18}$) | Minor = 26 | Minor = 27 | Minor = 28 | Minor = 29 |
| 11111...11110 | 0111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 |
| 11111...11101 | 1111 1111 1111 1111 | 0111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 |
| 11111...11011 | 1011 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 |
| 11111...10111 | 1111 1111 1111 1111 | 1011 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 |
| ... | ... | ... | ... | ... |
| 10111...11111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1110 |
| 01111...11111 | 1111 1111 1111 1111 | 1111 1111 1111 1111 | 1111 1111 1111 1110 | 1111 1111 1111 1111 |

```
LutContent getLutContent(int N_CR, int Y_CR, int X_T, int X_S, int Y_S, String LutType) {
  int Block = 0                              // CLB Block
  int max_row = N_CR/2 + N_CR%2              // N_CR is the number of clock regions
  int Top = (Y_CR > rows -1) ? 0 : 1         // Top / Bottom
  int Row = (Y_CR > rows -1) ? Y_CR - max_row : max_row - Y_CR - 1
  int Column = X_T
  int Minor = (X_S % 2 == 1) ? 26 : 32       // Odd (L) Slice/ Even (L/M) Slice
  int Word = (LutType == A || B) ? (Y_S % 50) * 2 : 1 + (Y_S % 50) * 2
  if (Word >= 50) Word++
  Range Bits = (LutType == A || C) ? [15:0] : [31:16]
  LutContent result
  result[63:48] = readFrame(Block, Top, Row, Column, Minor + 0, Word, Bits)
  result[47:32] = readFrame(Block, Top, Row, Column, Minor + 1, Word, Bits)
  result[31:16] = readFrame(Block, Top, Row, Column, Minor + 2, Word, Bits)
  result[15:0]  = readFrame(Block, Top, Row, Column, Minor + 3, Word, Bits)
  return result
}
```

Fig. 6. Algorithm that gets the LUT contents from the CM.

contents. As nearly all of the CM contents for this simple design are set to '0', this particular zero-cold pattern enables the easy location of the LUT contents within the CM, as shown in Table. II.

Similar experiments were performed for the rest of LUT BELs. Then, all these experiments were repeated for the other Slice ($X_{S+1}, Y_T$) within the same CLB Tile. After that, several similar experiments were carried out with Slices across the whole device to derive a generic formula for locating the LUT contents within the CM depending on the LUT's location within the device's architecture. For the sake of completeness, Fig. 6 lists the pseudocode that maps the LUT contents to the bits of the CM.

By using this directed mapping between the inputs of a LUT6 macrocell and a $6LUT$ BEL pins (*LOCK_PINS* attribute), it is possible to derive the relationship of each bit in its contents (*INIT* attribute) with the bits obtained from CM by the proposed algorithm. This precise mapping, which differs for LUTs located in L and M Slices, is listed in Table III.

However, this precise pin assignment is just one of the 720 possible assignments for LUT6 macrocells. Furthermore, as previously discussed, some BEL pins may be unused when implementing Boolean functions with less than 6 variables. Accordingly, it is necessary to recompute the bit mapping shown in Table III to account for the unused pins and the specific pin assignment by following these steps:

1) Rename the ($A6,\cdots,A0$) columns to match their associated macrocell input ($I$).
2) Remove all the entries that will never be accessed: unused pins are assumed to be tied to '1', except $A6$ that is tied to '0' when the LUT's output is $O6$.

TABLE III. Mapping of LUT6 contents to CM bits read by the proposed algorithm.

| LUT-BEL mapping | | | | | | INIT bit index | LutContent bit | |
|---|---|---|---|---|---|---|---|---|
| $I_5$ $A_6$ | $I_4$ $A_5$ | $I_3$ $A_4$ | $I_2$ $A_3$ | $I_1$ $A_2$ | $I_0$ $A_1$ | | Slice L | Slice M |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 63 | 31 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 47 | 15 |
| 0 | 0 | 0 | 0 | 1 | 0 | 2 | 62 | 30 |
| 0 | 0 | 0 | 0 | 1 | 1 | 3 | 46 | 14 |
| 0 | 0 | 0 | 1 | 0 | 0 | 4 | 61 | 29 |
| 0 | 0 | 0 | 1 | 0 | 1 | 5 | 45 | 13 |
| 0 | 0 | 0 | 1 | 1 | 0 | 6 | 60 | 28 |
| 0 | 0 | 0 | 1 | 1 | 1 | 7 | 44 | 12 |
| 0 | 0 | 1 | 0 | 0 | 0 | 8 | 15 | 63 |
| 0 | 0 | 1 | 0 | 0 | 1 | 9 | 31 | 47 |
| 0 | 0 | 1 | 0 | 1 | 0 | 10 | 14 | 62 |
| 0 | 0 | 1 | 0 | 1 | 1 | 11 | 30 | 46 |
| 0 | 0 | 1 | 1 | 0 | 0 | 12 | 13 | 61 |
| 0 | 0 | 1 | 1 | 0 | 1 | 13 | 29 | 45 |
| 0 | 0 | 1 | 1 | 1 | 0 | 14 | 12 | 60 |
| 0 | 0 | 1 | 1 | 1 | 1 | 15 | 28 | 44 |
| 0 | 1 | 0 | 0 | 0 | 0 | 16 | 59 | 27 |
| 0 | 1 | 0 | 0 | 0 | 1 | 17 | 43 | 11 |
| 0 | 1 | 0 | 0 | 1 | 0 | 18 | 58 | 26 |
| 0 | 1 | 0 | 0 | 1 | 1 | 19 | 42 | 10 |
| 0 | 1 | 0 | 1 | 0 | 0 | 20 | 57 | 25 |
| 0 | 1 | 0 | 1 | 0 | 1 | 21 | 41 | 9 |
| 0 | 1 | 0 | 1 | 1 | 0 | 22 | 56 | 24 |
| 0 | 1 | 0 | 1 | 1 | 1 | 23 | 40 | 8 |
| 0 | 1 | 1 | 0 | 0 | 0 | 24 | 11 | 59 |
| 0 | 1 | 1 | 0 | 0 | 1 | 25 | 27 | 43 |
| 0 | 1 | 1 | 0 | 1 | 0 | 26 | 10 | 58 |
| 0 | 1 | 1 | 0 | 1 | 1 | 27 | 26 | 42 |
| 0 | 1 | 1 | 1 | 0 | 0 | 28 | 9 | 57 |
| 0 | 1 | 1 | 1 | 0 | 1 | 29 | 25 | 41 |
| 0 | 1 | 1 | 1 | 1 | 0 | 30 | 8 | 56 |
| 0 | 1 | 1 | 1 | 1 | 1 | 31 | 24 | 40 |

| LUT-BEL mapping | | | | | | INIT bit index | LutContent bit | |
|---|---|---|---|---|---|---|---|---|
| $I_5$ $A_6$ | $I_4$ $A_5$ | $I_3$ $A_4$ | $I_2$ $A_3$ | $I_1$ $A_2$ | $I_0$ $A_1$ | | Slice L | Slice M |
| 1 | 0 | 0 | 0 | 0 | 0 | 32 | 55 | 23 |
| 1 | 0 | 0 | 0 | 0 | 1 | 33 | 39 | 7 |
| 1 | 0 | 0 | 0 | 1 | 0 | 34 | 54 | 22 |
| 1 | 0 | 0 | 0 | 1 | 1 | 35 | 38 | 6 |
| 1 | 0 | 0 | 1 | 0 | 0 | 36 | 53 | 21 |
| 1 | 0 | 0 | 1 | 0 | 1 | 37 | 37 | 5 |
| 1 | 0 | 0 | 1 | 1 | 0 | 38 | 52 | 20 |
| 1 | 0 | 0 | 1 | 1 | 1 | 39 | 36 | 4 |
| 1 | 0 | 1 | 0 | 0 | 0 | 40 | 7 | 55 |
| 1 | 0 | 1 | 0 | 0 | 1 | 41 | 23 | 39 |
| 1 | 0 | 1 | 0 | 1 | 0 | 42 | 6 | 54 |
| 1 | 0 | 1 | 0 | 1 | 1 | 43 | 22 | 38 |
| 1 | 0 | 1 | 1 | 0 | 0 | 44 | 5 | 53 |
| 1 | 0 | 1 | 1 | 0 | 1 | 45 | 21 | 37 |
| 1 | 0 | 1 | 1 | 1 | 0 | 46 | 4 | 52 |
| 1 | 0 | 1 | 1 | 1 | 1 | 47 | 20 | 36 |
| 1 | 1 | 0 | 0 | 0 | 0 | 48 | 51 | 19 |
| 1 | 1 | 0 | 0 | 0 | 1 | 49 | 35 | 3 |
| 1 | 1 | 0 | 0 | 1 | 0 | 50 | 50 | 18 |
| 1 | 1 | 0 | 0 | 1 | 1 | 51 | 34 | 2 |
| 1 | 1 | 0 | 1 | 0 | 0 | 52 | 49 | 17 |
| 1 | 1 | 0 | 1 | 0 | 1 | 53 | 33 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 54 | 48 | 16 |
| 1 | 1 | 0 | 1 | 1 | 1 | 55 | 32 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 56 | 3 | 51 |
| 1 | 1 | 1 | 0 | 0 | 1 | 57 | 19 | 35 |
| 1 | 1 | 1 | 0 | 1 | 0 | 58 | 2 | 50 |
| 1 | 1 | 1 | 0 | 1 | 1 | 59 | 18 | 34 |
| 1 | 1 | 1 | 1 | 0 | 0 | 60 | 1 | 49 |
| 1 | 1 | 1 | 1 | 0 | 1 | 61 | 17 | 33 |
| 1 | 1 | 1 | 1 | 1 | 0 | 62 | 0 | 48 |
| 1 | 1 | 1 | 1 | 1 | 1 | 63 | 16 | 32 |

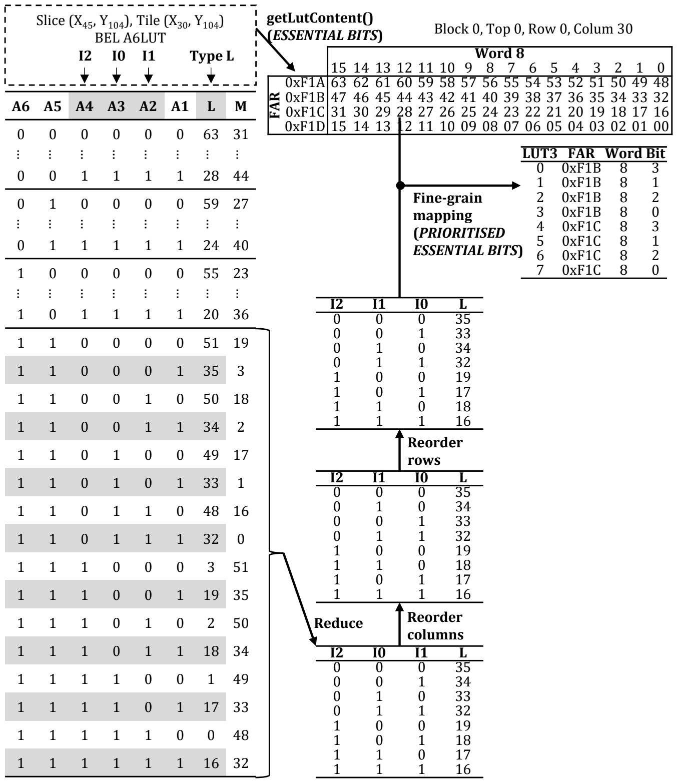
3) Remove the columns corresponding to the unused pins.
4) Reorder columns $I$ by descending index (left to right).
5) Reorder rows by ascending value of $I$ vector (top to bottom).

As a result, the columns L and M in the reduced and reordered Table III contain the bit indexes within the LUT contents retrieved from the CM that directly correspond to the *INIT* bits of the mapped LUT macrocell in ascending order. Fig. 7 displays the procedure followed for a given LUT to get the exact location of all its content bits in the CM.

In this way, it is now possible to restrict the number of fault injection experiments related to LUTs to those essential bits that may potentially affect the behaviour of the design, the so-called *prioritised essential bits* (see Fig. 1). The higher the underused LUT pins, the higher the expected reduction.

## *B. Dynamic Analysis of LUTs*

SEUs impacting any prioritised essential bit change the structure of the implemented design and, as such, may affect its functionality. However, it is up to the particular workload


Slice ($X_{45}$, $Y_{104}$), Tile ($X_{30}$, $Y_{104}$)
BEL A6LUT
I2
I0
I1
Type L
getLutContent()
(*ESSENTIAL BITS*)
Block 0, Top 0, Row 0, Colum 30
Fine-grain mapping (*PRIORITISED ESSENTIAL BITS*)
Reorder rows
Reorder columns
Reduce

| A6 | A5 | A4 | A3 | A2 | A1 | L | M |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 63 | 31 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 0 | 0 | 1 | 1 | 1 | 1 | 28 | 44 |
| 0 | 1 | 0 | 0 | 0 | 0 | 59 | 27 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 0 | 1 | 1 | 1 | 1 | 1 | 24 | 40 |
| 1 | 0 | 0 | 0 | 0 | 0 | 55 | 23 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1 | 0 | 1 | 1 | 1 | 1 | 20 | 36 |
| 1 | 1 | 0 | 0 | 0 | 0 | 51 | 19 |
| 1 | 1 | 0 | 0 | 0 | 1 | 35 | 3 |
| 1 | 1 | 0 | 0 | 1 | 0 | 50 | 18 |
| 1 | 1 | 0 | 0 | 1 | 1 | 34 | 2 |
| 1 | 1 | 0 | 1 | 0 | 0 | 49 | 17 |
| 1 | 1 | 0 | 1 | 0 | 1 | 33 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 48 | 16 |
| 1 | 1 | 0 | 1 | 1 | 1 | 32 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 3 | 51 |
| 1 | 1 | 1 | 0 | 0 | 1 | 19 | 35 |
| 1 | 1 | 1 | 0 | 1 | 0 | 2 | 50 |
| 1 | 1 | 1 | 0 | 1 | 1 | 18 | 34 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 49 |
| 1 | 1 | 1 | 1 | 0 | 1 | 17 | 33 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 48 |
| 1 | 1 | 1 | 1 | 1 | 1 | 16 | 32 |

| FAR | Word 8 | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0xF1A | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 |
| 0xF1B | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 0xF1C | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| 0xF1D | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

| LUT3 | FAR | Word | Bit |
|---|---|---|---|
| 0 | 0xF1B | 8 | 3 |
| 1 | 0xF1B | 8 | 1 |
| 2 | 0xF1B | 8 | 2 |
| 3 | 0xF1B | 8 | 0 |
| 4 | 0xF1C | 8 | 3 |
| 5 | 0xF1C | 8 | 1 |
| 6 | 0xF1C | 8 | 2 |
| 7 | 0xF1C | 8 | 0 |

| I2 | I1 | I0 | L |
|---|---|---|---|
| 0 | 0 | 0 | 35 |
| 0 | 0 | 1 | 33 |
| 0 | 1 | 0 | 34 |
| 0 | 1 | 1 | 32 |
| 1 | 0 | 0 | 19 |
| 1 | 0 | 1 | 17 |
| 1 | 1 | 0 | 18 |
| 1 | 1 | 1 | 16 |

| I2 | I1 | I0 | L |
|---|---|---|---|
| 0 | 0 | 0 | 35 |
| 0 | 1 | 0 | 34 |
| 0 | 0 | 1 | 33 |
| 0 | 1 | 1 | 32 |
| 1 | 0 | 0 | 19 |
| 1 | 1 | 0 | 18 |
| 1 | 0 | 1 | 17 |
| 1 | 1 | 1 | 16 |

| I2 | I0 | I1 | L |
|---|---|---|---|
| 0 | 0 | 0 | 35 |
| 0 | 0 | 1 | 34 |
| 0 | 1 | 0 | 33 |
| 0 | 1 | 1 | 32 |
| 1 | 0 | 0 | 19 |
| 1 | 0 | 1 | 18 |
| 1 | 1 | 0 | 17 |
| 1 | 1 | 1 | 16 |

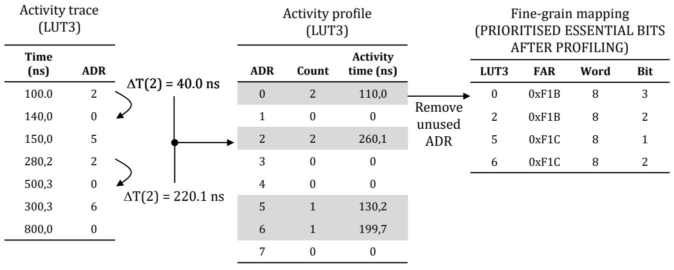
Fig. 7. Example of fine-grain mapping a LUT3 macrocell contents in the CM. LUT entries used are shaded in grey.

and incoming inputs to propagate this error through the design. For instance, if a CM bit corresponding to entry 0 of a specific LUT BEL is flipped (from '0' to '1' or from '1' to '0') and this entry is never accessed during operation, this fault will never lead to failure. This fact provides an opportunity to reduce further the set of prioritised essential bits attending to the dynamic behaviour of the implemented design.

State-of-the-art simulation tools, like Siemens ModelSim/QuestaSim, offer profiling functionalities to monitor the switching activity of selected signals in a design. Hence, if the workload of a given design is known, it could be possible to profile the switching activity of all LUTs' interfaces using an implementation-level (timing-accurate) simulation model.

Fig. 8 continues the example presented in Fig. 7 to reduce the set of prioritised essential bits obtained after the static analysis of the design using profiling. Profiling table entries with zero activity time (or a *Count* value of 0) indicate those LUT bits that have remained inactive during the workload execution and, thus, can be ignored during fault injection. In this case, the workload does not use 4 out of 8 bits, reducing the prioritised essential bits for that LUT of 50%.

It must be noted that when implementing two Boolean functions that do not share inputs, each *INIT* bit of the profiled LUT maps onto several CM cells. Thus, to determine which CM cell is active at each instant, it is necessary to consider the switching activity of both combined LUT primitives in the same activity trace.



| Time (ns) | ADR |
|---|---|
| 100.0 | 2 |
| 140,0 | 0 |
| 150,0 | 5 |
| 280,2 | 2 |
| 500,3 | 0 |
| 300,3 | 6 |
| 800,0 | 0 |

| ADR | Count | Activity time (ns) |
|---|---|---|
| 0 | 2 | 110,0 |
| 1 | 0 | 0 |
| 2 | 2 | 260,1 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |
| 5 | 1 | 130,2 |
| 6 | 1 | 199,7 |
| 7 | 0 | 0 |

| LUT3 | FAR | Word | Bit |
|---|---|---|---|
| 0 | 0xF1B | 8 | 3 |
| 2 | 0xF1B | 8 | 2 |
| 5 | 0xF1C | 8 | 1 |
| 6 | 0xF1C | 8 | 2 |

Fig. 8. Example of profiling the switching activity of the macrocell from Fig. 7 to reduce the set of prioritised essential bits. LUT entries used are shaded in grey.

## IV. Case Study

As previously mentioned, mapping the complete set of CM bits of an FPGA is very challenging and time-consuming, so the question is whether this proposal pays in terms of experimental time reduction against simply deploying a fault injection campaign on all essential bits automatically identified by the manufacturer development tools. This case study shows that the approach pays even when just considering the bits of the CM devoted to the configuration of the design combinational logic.

The considered case study integrates the implementation of three soft-core processors (AVR [17], MC8051 [18], and Microblaze [19]) onto a Xilinx Zynq-7000 SoC FPGA (xc7z020) using Xilinx Vivado 2018.3 design suite. It enables the comparison of different approaches for estimating the sensitivity of a design to SEUs, in particular: i) conservatively (blindly) assuming all CM bits are essential, ii) using Xilinx essential bits information, iii) using the prioritised essential bits obtained after the proposed static and dynamic analysis, and iv) quantifying the exact number of critical bits through FPGA-based fault injection experiments.

The publicly available DAVOS toolkit [20] has been used to emulate the occurrence of upsets (single bit-flips) in each cell of the CM before executing the workload and to check whether this bit is critical (results differ from the fault-free run) or not. Before each fault injection experiment, the CM content and the design context reset to their initial state. All the designs under test run the same synthetic matrix multiplication workload.

The number of essential bits estimated after the considered approaches is listed in Table IV. As it can be seen, the CM of the selected device holds roughly 23.7 Mbit. Suppose all of them are considered potentially critical under an upset rate of 74 FIT/Mbit (as reported by Xilinx reliability report [10] for the selected FPGA family). In that case, this will lead to a failure rate estimation of 1753.8 FIT. Even if taking into account the typical device vulnerability factor (percentage of critical bits) of 5% to 10% [10], this estimation will range between 87.7 FIT and 175.4 FIT.

Analysing the Xilinx essential bits information (*.ebc file produced by Vivado) can provide more accurate estimations. Table IV reports 474 Kbit, 342 Kbit, and 351 Kbit of essential

TABLE IV. Total (a) and LUT-related (b) essential bits obtained by using a blind approach (all device CM), the manufacturer (Xilinx essential bits), and the proposed static and dynamic analysis (prioritised essential bits).

a) Total number of bits

| **Design** | **Blind approach (All CM bits)** | **Xilinx (Essential bits)** | **Static and dynamic analyses (Prioritised Essential bits)** | |
|---|---|---|---|---|
| | Kbits | Kbits | Kbits | Reduction |
| MC8051 | 23745.5 | 474.6 | 383.2 | 19% |
| AVR | 23745.5 | 342.9 | 295.2 | 14% |
| Microblaze | 23745.5 | 351.7 | 296.3 | 16% |

b) Bits related to LUTs

| **Design** | **Xilinx (Essential bits)** | | **Static and dynamic analyses (Prioritised Essential bits)** | | | |
|---|---|---|---|---|---|---|
| | | | Static analysis | | Dynamic analysis | |
| | Kbits | % of total | Kbits | Reduction | Kbits | Reduction |
| MC8051 | 144.4 | 30% | 113.8 | 21% | 52.9 | 63% |
| AVR | 98.6 | 29% | 78.6 | 20% | 50.9 | 48% |
| Microblaze | 71.9 | 20% | 32.2 | 55% | 16.5 | 77% |

TABLE V. Estimated critical bits using FPGA-based fault injection when targeting the essential bits reported by Xilinx Vivado, and the prioritised bits obtained after the proposed static and dynamic analysis.

| | **Xilinx Essential bits** | | **Static analysis** | | | | **Dynamic analysis** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Critical bits (Kbits) | Experimentation (hours) | Critical bits (Kbits) | Mapping overhead (hour) | Experimentation including overhead (hours) | Speed-up | Critical bits (Kbits) | Profiling overhead (hours) | Experimentation including overhead (hours) | Speed-up |
| MC8051 | 56.0 | 3.42 | 56.0 | 0.02 | 3.19 | 1.07 | 56.0 | 0.19 | 2.96 | 1.16 |
| AVR | 56.9 | 2.02 | 56.9 | 0.02 | 1.90 | 1.06 | 56.9 | 0.07 | 2.20 | 1.12 |
| Microblaze | 75.9 | 5.33 | 75.9 | 0.02 | 4.80 | 1.11 | 75.9 | 0.14 | 4.67 | 1.14 |

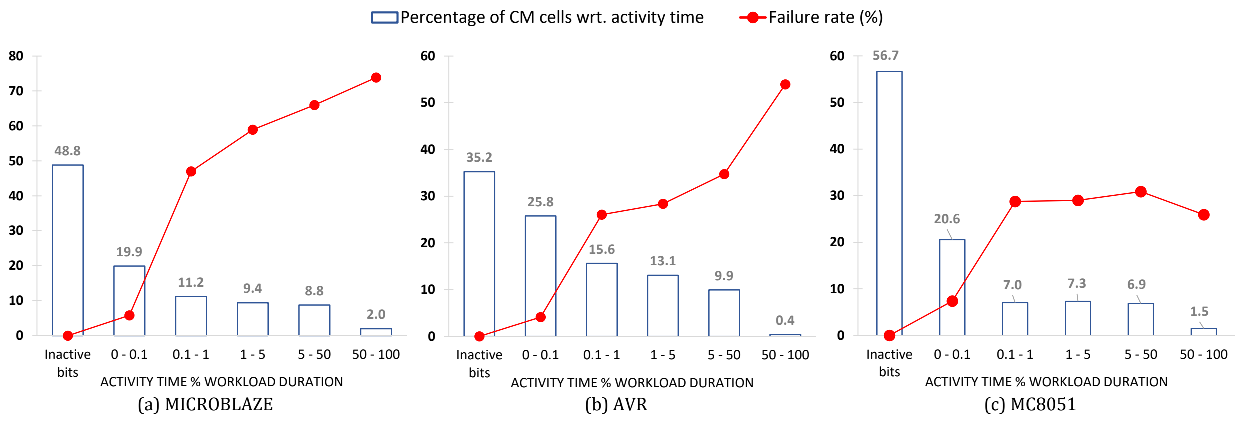


Fig. 9. Percentage of LUT-specific CM cells with respect to their profiled activity time and failure rate.

bits for MC8051, AVR, and Microblaze, respectively. This significantly refines the initial failure rate estimation, reducing it to just 35.1 FIT (MC8051), 25.3 FIT (AVR), and 26.0 FIT (Microblaze), well below the worst-case scenario computed.

Between 20% and 30% of these essential bits (144 Kbit (MC8051), 99 Kbit (AVR), and 72 Kbit (Microblaze)) are attributed to the LUT content. Applying the proposed LUT mapping algorithm reduces this amount of LUT-specific essential bits between 20% (AVR) and 55% (Microblaze). Furthermore, profiling has shown that between 35.2% (AVR) and 56.7% (MC8051) of LUT-specific CM cells remain inactive during the execution of the considered workload. By combining both proposals, the initial estimation of LUT-specific essential bits provided by Vivado is reduced between 48% (AVR) and 77% (Microblaze).

Accordingly, the set of prioritised essential bits is between 14% and 19% smaller than that provided by Vivado. Thus, the estimated failure rates are reduced to 28.3 FIT (MC8051), 21.8 FIT (AVR), and 21.9 FIT (Microblaze). Although this refinement may seem not a vast improvement, it effectively narrows down the worst-case scenario without running a single fault injection experiment.

After applying the proposed static and dynamic analyses, all the essential bits of the designs were exhaustively targeted by fault injection experiments to estimate each design's exact number of critical bits. The results, summarised in Table V, show that the actual number of critical bits ranges between 75.9 Kbit (Microblaze) and 56.0 Kbit (MC8051). This leads to an even more refined estimation of the failure rate: 4.1 FIT (MC8051), 4.2 FIT (AVR), and 5.6 FIT (Microblaze). These experiments have confirmed that the occurrence of upsets in those essential bits reported by Vivado and excluded from the set of prioritised essential bits never leads to a failure. Therefore, they are not critical and can be safely disregarded to reduce the number of faults injected and speed up experimentation by 1.12 to 1.16.

It is to note that the overhead introduced by the static analysis is negligible and constant, as it depends on the size of the target device and not the design. Accordingly, it should always be applied to speed-up fault injection campaigns (between 1.06 and 1.11 for this case study) at a negligible cost. However, the dynamic analysis (profiling) benefits are not so clear. On the one hand, the simulation of implementation-level models is time-consuming and may result in significant overhead. Accordingly, it will depend on each particular design and workload whether the simulation time is longer than the time saved by reducing the number of essential bits. On the other hand, Fig. 9 reveals that the criticality of LUT-specific CM cells seems to increase as their activity time does. Therefore, activity profiles could indicate those CM cells with the highest probability of being critical even without costly fault injection experiments. Likewise, activity profiles could guide fault injection experiments to detect the highest possible number of critical CM cells under a limited experimental time.

## V. Conclusions and Future Work

As the potential benefits of FPGA-based designs for critical applications and their complexity increase, the need for new techniques to estimate their robustness also does. However, speeding up injection campaigns concerning what is currently provided by the design of experiments, parallelisation, and statistical fault injection is quite challenging.

This paper delves into the FPGA architecture to determine which combinational logic elements are used by the considered design (mapping through static analysis) and which are exercised by the workload (profiling through dynamic analysis). This reversing approach identifies a set of *prioritised essential bits* that limits the potential fault injection targets to consider for experimentation.

Results show that this work pays with respect to exhaustively injecting faults in all design essential bits. In all considered processor implementations, the number of bits to consider is reduced by up to 19%, whereas the subsequent fault injection campaign runs 1.16 times faster without loss of accuracy. Even though the savings may not be vast, the cost of applying the proposed static analysis is negligible. Although conditioned by the considered workload, the dynamic analysis remains of interest to identify critical CM cells requiring more protection.

We plan to continue with this research by testing the approach using a more representative set of benchmark circuits and enlarging its applicability to reverse the part of the CM devoted to the interconnection (routing) of the FPGA fabric elements. It must be noted that routing resources account for around 75% of the CM content of an FPGA. So, considering the speed up resulting from analysing just the combinational logic of the FPGA, this future investigation has a great potential not only to accelerate (even more) fault injection campaigns but also to exploit the information issued from the static and dynamic analysis to more precisely identify dependability bottlenecks and define more effective protection mechanisms.

## References

[1] R. C. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[2] V. Vlagkoulis, A. Sari, G. Antonopoulos, M. Psarakis, A. Tavoularis, G. Furano, C. Boatella-Polo, C. Poivey, V. Ferlet-Cavrois, M. Kastriotou, P. F. Martinez, and R. G. Alía, "Configuration Memory Scrubbing of SRAM-Based FPGAs Using a Mixed 2-D Coding Technique," *IEEE Transactions on Nuclear Science*, vol. 69, no. 4, pp. 871–882, 2022.

[3] H. Quinn and P. Graham, "Terrestrial-based radiation upsets: a cautionary tale," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005, pp. 193–202.

[4] C. Hu and S. Zain, "Nseu mitigation in avionics applications," *Xilinx Application Note XAPP1073 v1. 0*, pp. 1–12, 2010.

[5] C. Bolchini, A. Miele, and M. D. Santambrogio, "Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*. IEEE, 2007, pp. 87–95.

[6] Z. Gao, H. Zhang, Y. Yao, J. Xiao, S. Zeng, G. Ge, Y. Wang, A. Ullah, and P. Reviriego, "Soft Error Tolerant Convolutional Neural Networks on FPGAs With Ensemble Learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 291–302, 2022.

[7] Y. Ichinomiya, S. Tanoue, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "Improving the robustness of a softcore processor against seus by using tmr and partial reconfiguration," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2010, pp. 47–54.

[8] R. Glein, F. Rittner, A. Becher, D. Ziener, J. Frickel, J. Teich, and A. Heuberger, "Reliability of space-grade vs. cots sram-based fpga in n-modular redundancy," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2015, pp. 1–8.

[9] N. T. Nguyen, E. Cetin, and O. Diessel, "Scheduling voter checks to detect configuration memory errors in fpga-based tmr systems," in *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2017, pp. 1–4.

[10] Xilinx Inc., "Device Reliability Report, First Half 2018, UG116 (v10.9)," 2018.

[11] S. Di Carlo, P. Prinetto, D. Rolfo, and P. Trotta, "A fault injection methodology and infrastructure for fast single event upsets emulation on xilinx sram-based fpgas," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2014, pp. 159–164.

[12] Xilinx Inc., "7 Series FPGAs Configuration. UG470 (v1.13.1)," 2018.

[13] ——, "7 Series FPGAs Configurable Logic Block. UG474 (v1.8)," 2016.

[14] K. D. Pham, E. Horta, and D. Koch, "Bitman: A tool and api for fpga bitstream manipulations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 894–897.

[15] L. A. Cardona and C. Ferrer, "Ac_icap: a flexible high speed icap controller," *International Journal of Reconfigurable Computing*, vol. 2015, p. 15, 2015.

[16] M. Jeong, J. Lee, E. Jung, Y. H. Kim, and K. Cho, "Extract lut logics from a downloaded bitstream data in fpga," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[17] J. Sauermann, "How to design your own CPU on FPGAs with VHDL," 2010. [Online]. Available: https://github.com/freecores/cpu_lecture

[18] Oregano Systems GmbH, "MC8051 IP Core, Synthesizeable VHDL Microcontroller IP-Core, User Guide (V 1.2), 2013," 2013.

[19] Xilinx Inc., "Microblaze processor reference guide, ug984," 2019.

[20] I. Tuzov, D. de Andrés, and J. C. Ruiz, "DAVOS: EDA Toolkit for Dependability Assessment, Verification, Optimisation and Selection of Hardware Models," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 322–329.