A Technical Guide for the RASP-FIT Tool

Abdul Rafay Khatri
Department of Computer Architecture and System Programming,
University of Kassel, Kassel, Germany.

***Abstract*—Fault injection tools are designed to serve various purposes, such as validate the design under test concerning reliability requirements, find sensitive/critical locations that require error mitigation, determine the expected circuit response in the existence of faults. Fault Simulation/Emulation (S/E) applications are involved in Field Programmable Gate Array (FPGA) based design's verification and simulation at the Hardware Description Languages (HDL) code level. A tool is developed, named RASP-FIT, to perform code modification of FPGA designs, testing of such designs, and finding the sensitive area of designs. This tool works on the FPGA designs written in Verilog HDL at various abstraction levels, gate, data-flow and behavioural levels. This paper presents a technical aspect and the user-guide for the proposed tool in detail, which includes generation of the standalone application (an executable file of the tool for Windows operating system) and installation method.**

Keywords*—*Code-modifier; fault injection; FPGA designs; fault injection tool; Verilog HDL

I. Introduction

Dependability is the study of error and failure. A robust method that allows assessing the reliability of a target system is the Fault Injection (FI) method. Therefore, fault injection technique can be defined as "the dependability validation technique that is performed in a controlled experiment for the System Under Test (SUT) and observed its response in the presence of faults" [1]. Fault injection technique is used to test the fault-tolerant mechanisms of a system when known faults take place, and evaluate in this way for their effectiveness. The primary goals of fault injection in the design process are validation and design aid.

In the validation process, fault injection is intended to test the mechanisms implemented by the system to achieve dependability (fault tolerance mechanisms) concerning the faults that they are injected during the fault injection campaign. The validation process through fault injection has two primary purposes: fault removal, which is based on the design verification, and fault forecasting, which depends on the assessment of the system. When the objective is fault removal, a qualitative analysis is performed to check if the fault tolerance mechanisms are suitable regarding dependability requirements of the system. On the other hand, in the case of fault forecasting, fault injection is intended to perform quantitative evaluation (with a probabilistic approach), and the coverage of the fault tolerance mechanisms evaluated by the system [2], [3]. In the design aid, fault injection experiments are executed at several steps of the development flow. Results of fault injection are then used to initiate an iterative process that allows improving the test procedures and the fault tolerance mechanisms of the system being exploited.

Fault injection is a useful technique to estimate design characteristics such as reliability, safety, and fault coverage. The method consists of intentionally injecting faults into the device under test and observing the behaviour of faults/errors [1], [4]. Fault injection covers many fundamentals objectives:

1) Validate the design under test concerning reliability.
2) Detect critical areas require mitigation of errors.
3) Determine the expected circuit behaviour in the faulty environment.

The fault injection environment set-up consists of fault location, time of injection, duration of active faults, and the input data for the system.

Hardware Description Language (HDL) has been involved in designing the digital system during the last many years for Field Programmable Gate Array (FPGA) and Application Specified Integrated Circuits (ASIC). Testing and other fault simulation applications can now be applied directly at the HDL code level. Due to this, the gap between the tools and methods used by design and test engineers is reduced [5]. The HDL represents a higher abstraction level in the design flow. Testing should be carried out at lower abstraction levels to obtain the best responses. However, an HDL model at the behavioural level can be simulated to produce useful circuit response vectors for test purposes efficiently. The SUT and its hardware simulation model at the code level can be analysed for testability method.

This paper describes the technical user-guide for a novel fault injection tool and explains the way the user can use this tool for Verilog HDL at any abstraction level. The tool is named RASP-FIT (RechnerArchitektur und SystemProgrammierung-Fault Injection Tool) after the German name of the department in which it is developed. This fault injection tool can be used to modify the design for the Automatic Test Pattern Generation (ATPG) and other fault simulation applications. This tool is programmed in Matlab in a Graphical User Interface Development Environment (GUIDE). This paper presents the method to develop a standalone application, its installation on the computer without having a Matlab tool. Also, it describes the way to use this tool step by step.

The organisation of the paper is as follows: Section II describes the various fault injection techniques and tools for FPGA-based designs. Section III introduces the structure of the RASP-FIT in Matlab along with the method to build and install the standalone application for the user of the tool. The working procedure is also described in the section. Result and discussion for an example design using the RASP-FIT tool are mentioned in Section IV. In the end, Section V concludes the paper.

II. Related Work

Fault injection and fault simulation are typical methods to investigate the impact of a fault on a hardware/software system. Usually, fault injection is performed on abstract models of the system either to retrieve early results when no implementation is available, yet, or to speed up the run-time fast fault simulation of the specified models. Fault injection is mainly used to evaluate fault-tolerant mechanisms [5]. In the last few decades, fault injection has become a popular technique for experimentally determining dependability parameters of a system, such as a fault latency, fault propagation and fault coverage. Types of FI tools for FPGA in the literature are described in the sequel.

A. FI Techniques/Tools based on Simulation

Tools based on simulation involve the simulation model of the design under analysis. The errors or failures for the SUT are distributed with proposed mechanisms [6]. These techniques and tools are divided into two, i.e. Code-Modification (CM) and Simulator Command (SC):

1) Code-Modification technique:- This technique requires the modification of HDL code by adding some fault models such as stuck-at, bit-flip, mutant and saboteur.
2) Simulator Command technique:- In this type of tools and techniques, the particular simulator command is used to change the values of the signal or variable of HDL models available with the simulator (e.g. Modelsim, Xilinx ISIM).

The advantages are summarized below for the Simulation-Based Fault Injection (SBFI) tools [2], [1]:

- As simulation model is used instead of the actual hardware model hence there is no risk of damage.
- Being cost effective.
- Provides higher controllability and observability during FI experiments.
- Different fault models can be modelled with ease.
- Supporting any type of HDL code.

Most the SBFI tools in the literature, are available for the VHDL language, such as VERIFY [7], (MEFISTO-C, HEARTLESS, VFIT (VHDL-based Fault Injection Tool), FTI (Fault Tolerance Injection)) [8], [2], Full System simulator-based Fault Injection (FSFI) [9], etc. These tools are based on simulator command and code-modification techniques. Fault modelling is achieved by a saboteur, and mutant injection. Verilog Programming Language Interface (PLI) application is used in the test generation method during fault simulation applications [5]. The top-level design module can also be modified in some cases to achieve test generation and fault simulation applications with the help of simulator command technique, as presented in [10].

B. FI Techniques/Tools based on Emulation

The emulation-based fault injection tools are most often used with FPGA for speeding up the fault injection experiments to achieve a faster solution. Design from specifications to implementation takes several steps. Emulation-based FI tools are divided by following the stages of the design flow. The two main categories are given in the sequel shortly, i.e. instrumentation and reconfiguration.

1) Instrumentation technique:- This technique requires the modification of HDL code by adding some fault models such as stuck-at, bit-flip, mutant and saboteur in the system, netlist or other formats of the FPGA-based design.
2) Reconfiguration technique:- Reconfiguration or partial reconfiguration is the technique in which the configuration memory of the FPGA is modified or changed with some other logic to inject faults in the SUT.

The fault injection tools develop to work on the net-list obtained by the synthesis process are presented in [11], [12], [13]. Some tools work on the code level and modify the design by instrumentation technique are presented in [14], [15]. However, some hybrid techniques (simulation/emulation) can be achieved by combining two or more fault injection techniques as given in [16], [17], [18], [19]. HDL environment can generate a list of faults, and it is used for fault emulation/simulation of the SUT. The author compared the RASP-FIT tool with the work presented in [20] and found that the RASP-FIT tool uses three fault models (bit-flip and stuck-at (1/0)) provides better performance in testing and hardness analysis.

III. Structure of the RASP-FIT Tool

The RASP-FIT tool, with its Graphical User Interface (GUI) is developed in Matlab. The tool consists of three major functions, namely [21], [22]:

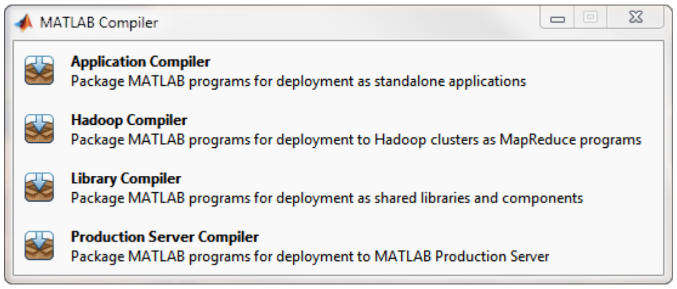
1) Fault Injection Analysis
2) Hardness Analysis
3) Static Compaction

All these functions are developed in Matlab under the function `RASP_FIT()`. It is a tabbed-based GUI as shown in Fig. 4, 5 and 6. Each tab performed certain specific functions and described in this work. To ease of use, a standalone Matlab GUI is developed for the proposed tool using the `deploytool` command [23]. This command generates the executable file for the RASP-FIT tool which can be installed on any computer containing Windows operating systems. The procedure for building a standalone application is given in the sequel.

A. Building a Standalone Tool

The standalone application helps the user of the tool to operate with ease. As described earlier, this tool is programmed in Matlab using the programmatic environment. To run the Matlab code, it is necessary to have a Matlab compiler installed on the computer. The standalone application can be run on any computer without Matlab. To run the RASP-FIT tool, a standalone app is generated. The procedure for creating a standalone application for the RASP-FIT device is described in the sequel.

1) Run the following command on the Matlab prompt.


MATLAB Compiler

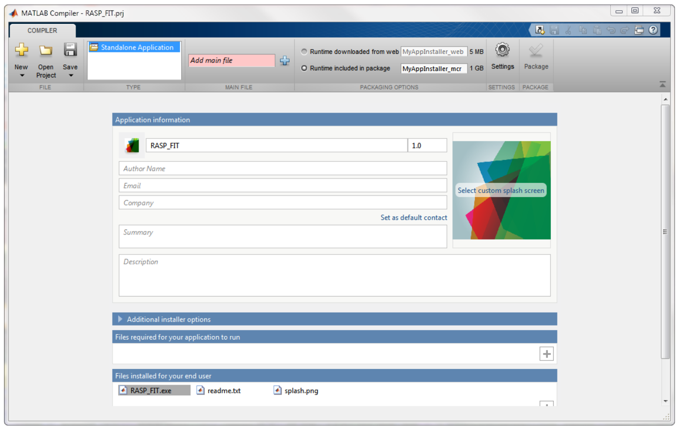
Application Compiler
Package MATLAB programs for deployment as standalone applications

Hadoop Compiler
Package MATLAB programs for deployment to Hadoop clusters as MapReduce programs

Library Compiler
Package MATLAB programs for deployment as shared libraries and components

Production Server Compiler
Package MATLAB programs for deployment to MATLAB Production Server

Fig. 1. Matlab compiler


MATLAB Compiler - RASP_FIT.prj

Application information
RASP_FIT 1.0
Author Name
Email
Company
Set as default contact
Summary
Description
Select custom splash screen
Additional installer options
Files required for your application to run
Files installed for your end user
RASP_FIT.exe readme.txt splash.png

Fig. 2. Matlab compiler window for creating project

```
>> deploytool
```

and press ENTER.

2) A window appears as shown in Fig. 1, select the first option “Application Compiler”.
3) After selecting the above option, another window appears, as shown in Fig. 2. Add the main code file and fill the required pieces of information. Select the option “RASP-FIT_Installer_web” and click on the “Package” option to start the process. After the process, the project is saved under the extension ‘*.prj’ and generates the executable file for the project.
4) When this process stops, the executable file is available in the folder named “for_redistribution”.

B. Installation Procedure

The stepwise procedure is explained to install the standalone application [24].

1) Open the RASP-FIT_Installer executable file located in the *for_redistribution* folder generated by the Matlab compiler.
2) Run the installer file (*.exe) by double click on it. The first window of the Fig. 3 is appeared.
3) Click on the Next option and it leads to the *Installation Options* page.
4) Set the folder location, check the box to generate short-cut to the desktop and click *Next*. See the second window of the Fig. 3.
5) Agree to the license agreement and click yes to it.
6) Click *Next* moves to the confirmation page and press the install button. It checks whether the Matlab runtime is installed in your computer or not. If needed, it also downloads and installs the Matlab runtime. See the third window of the Fig. 3.
7) When the installation is completed, the fourth window of Fig. 3 appears. Click finish and complete the installation.
8) Run the standalone application.

C. Working Procedure

After the successful installation, the user has to double click the RASP-FIT icon available on the desktop. At first, the RASP-FIT tool asks for the user-defined primitive file which is available with the tool by the provider. The user has to locate the file only, and graphical user interface for the tool is opened for use. As described earlier, the RASP-FIT tool performs three functions. The sequel portrays the details of each tab of the device and its related options.

1) Fault Injection Analysis: The second tab of the RASP-FIT tool is the fault injection analysis, as shown in Fig. 4. The user must provide three input for modifying the design for the fault injection analysis.

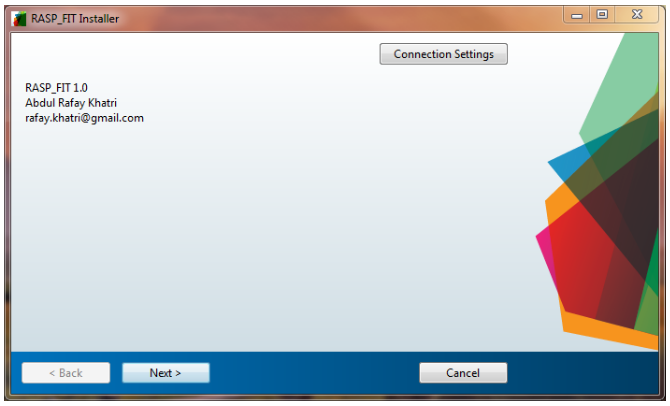
1) Synthesizable Verilog design file.
2) Select fault model for analysis from a drop-down menu.
3) Enter number of copies the user wants to generate with evenly distributed faults in them.

By clicking on the generate button, faulty modules and top module are produced and stored in a folder where the source file is located. The faulty modules are named (`moduleName_faultycopy1.v`, `moduleName_faultycopy2.v` and so on). The top file consists of fault injection testing logic. This logic contains the comparator logic, dynamic compaction scheme and memory declaration for storing the results of the comparisons, and it is stored under the name (`moduleName_top.v`) in the same folder. These modified designs help design and test engineers to perform fault simulation, digital testing and dependability analysis without much effort. Verilog HDL code modification techniques for each abstraction level are presented in [21], [25], [26]. Along with the faulty copies, the RASP-FIT also provides the number of copies generated, the number of faults per copies which is used to calculate the number of select port pins, the number of total defects injected in the design. In the end, it calculates the select port pins for faults per copy. Eq. 1 describes the method to calculate select port pins [27].

$$FS = \lceil log_2(F_{copy}) \rceil \tag{1}$$

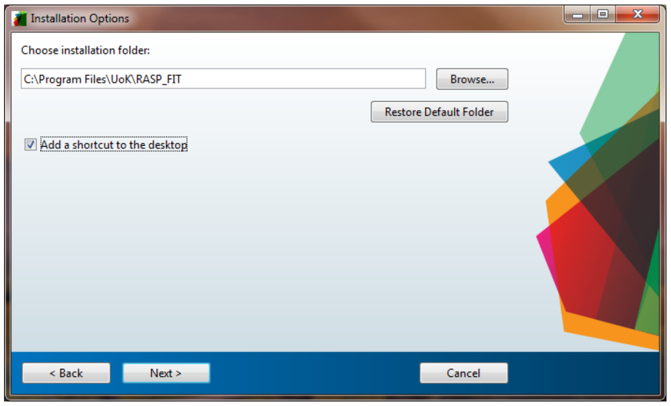
where F_{copy} denotes the number of faults injected per copy of the SUT and Fault Select (FS) is the number of select port pins.

2) Hardness Analysis: Hardness analysis is carried out for the SUT to find the characteristic of those faults which can be detected very often or rarely. The third tab of the RASP-FIT tool is for hardness analysis. It consists of three panels, namely, file merger section, an input data file section of hardness analysis and an input parameters required for



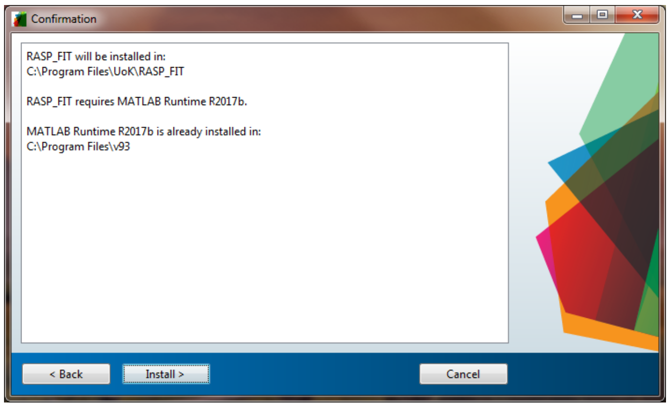
RASP_FIT 1.0
Abdul Rafay Khatri
rafay.khatri@gmail.com

Step 1: First window



Choose installation folder:
C:\Program Files\UoK\RASP_FIT
☑ Add a shortcut to the desktop

Step 2: Second window (installation folder)

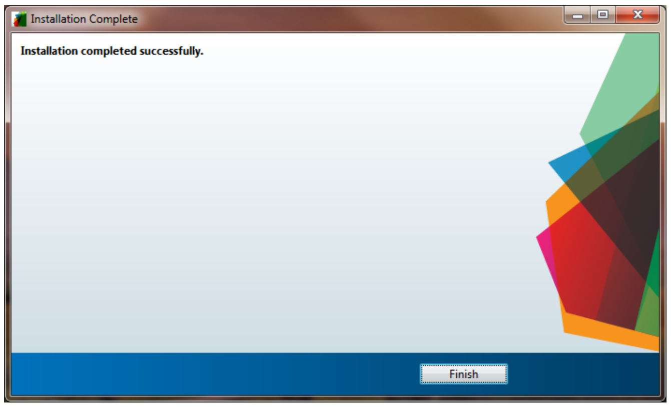


RASP_FIT will be installed in:
C:\Program Files\UoK\RASP_FIT

RASP_FIT requires MATLAB Runtime R2017b.

MATLAB Runtime R2017b is already installed in:
C:\Program Files\v93

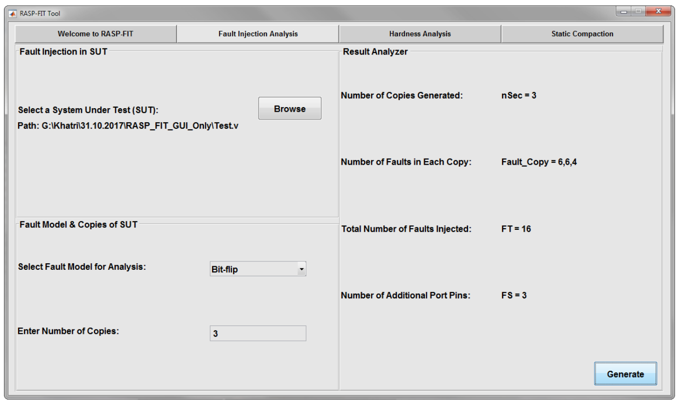
Step 3: Third window (confirmation window)



Installation completed successfully.

Step 4: Fourth window

Fig. 3. Installation procedure step by step

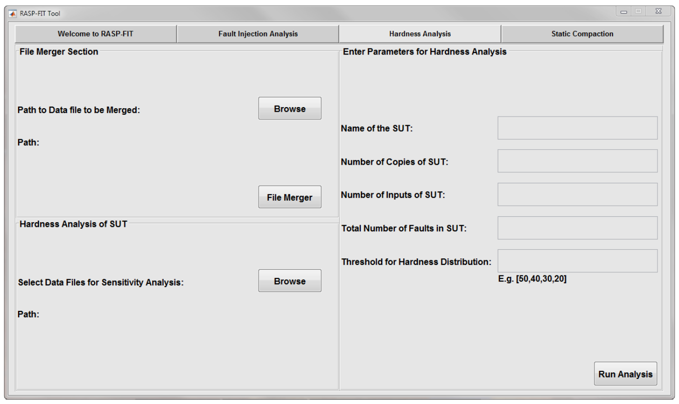


Fault Injection in SUT
Select a System Under Test (SUT):
Path: G:\Khatri\31.10.2017\RASP_FIT_GUI_Only\Test.v

Fault Model & Copies of SUT
Select Fault Model for Analysis: Bit-flip
Enter Number of Copies: 3

Result Analyzer
Number of Copies Generated: nSec = 3
Number of Faults in Each Copy: Fault_Copy = 6,6,4
Total Number of Faults Injected: FT = 16
Number of Additional Port Pins: FS = 3

Fig. 4. Fault injection analysis tab.



File Merger Section
Path to Data file to be Merged:
Path:

Hardness Analysis of SUT
Select Data Files for Sensitivity Analysis:
Path:

Enter Parameters for Hardness Analysis
Name of the SUT:
Number of Copies of SUT:
Number of Inputs of SUT:
Total Number of Faults in SUT:
Threshold for Hardness Distribution:
E.g. [50,40,30,20]

Fig. 5. Hardness analysis tab.

hardness analysis calculations. File merger is a program which takes many data files and merges them to produce one file. It is used if needed. When data files for hardness analysis are imported, and some parameter's information should be provided as inputs to perform hardness analysis. Fig. 5 shows the tab for hardness analysis. These inputs are:

- Name of the SUT: All hardness analysis results are saved under this name.
- Number of copies of SUT: This represents the number

of copies of SUT considered for the project.

- Number of inputs of SUT: Data is stored during experimentation as inputs and outputs. So for the removing of stored input patterns from the data, the user needs to provide this information.
- Total Number of Faults in SUT: This is a crucial parameter to calculate hardness analysis. This information provides the number of fault injection in the SUT during experiments.
- Threshold for Hardness Distribution: To divide the sensitive locations into hard to detect and most often detected, the user assign threshold values.

3) Static Compaction: Static compaction technique reduces the number of test vectors further after their generation. As static compaction techniques are not part of Test Pattern Generation (TPG), hence they do not change the TPG process. Therefore, It can be developed in any programming language or tool. The proposed static compaction algorithm is programmed in the Matlab, which needs few input parameters and data files from experiments. It calculates the fault coverage, compaction and reduces the number of test vectors. Fig. 6 shows the tab for static compaction. All the input parameters required to calculate compact test vectors and fault coverage are the same as that of hardness analysis except the second parameter. This parameter requires the number of fault models used in the test approach. The RASP-FIT uses three fault models at this stage of development. Hence, these parameters get value 3. Also, the user needs to provide three files, one for each fault model. In the next work, other fault models are also developed and added to the RASP-FIT tool easily.

It is seen that this RASP-FIT tool is straightforward, easy to use, and it does not require much computer skills to operate it. It validates our claims about the simplicity, ease of use and user-friendly tool.

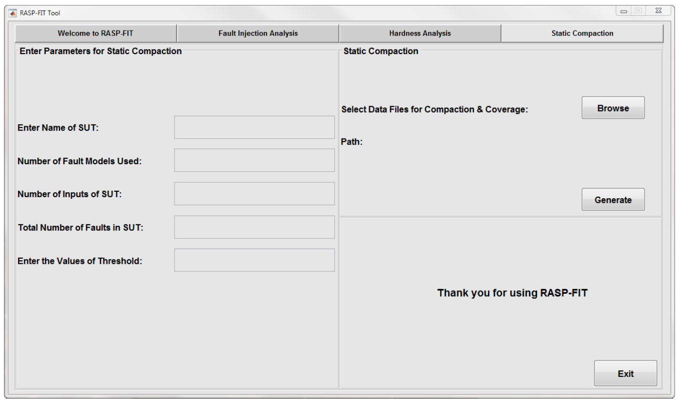

RASP-FIT Tool
Welcome to RASP-FIT
Fault Injection Analysis
Hardness Analysis
Static Compaction
Enter Parameters for Static Compaction
Enter Name of SUT:
Number of Fault Models Used:
Number of Inputs of SUT:
Total Number of Faults in SUT:
Enter the Values of Threshold:
Static Compaction
Select Data Files for Compaction & Coverage:
Browse
Path:
Generate
Thank you for using RASP-FIT
Exit

Fig. 6. Static compaction tab.

IV. Result and Discussion

After successful installation of the RASP-FIT tool on the host computer, the user can run it by double click the RASP-FIT icon. Firstly, the tool asks the user to locate the “user-defined-netlist.csv” file, which is provided with the executable file. This file contains the user-defined primitives and functions. There are two columns in the file separated by the semicolon ‘;’. The first column consists of the name of primitives or task and the second column contains the fault insertion location. When RASP-FIT is run, the contents of the file are read and added to the predefined respective libraries accordingly. For example, the user defines *FD* as a user-defined primitive for a flip-flop with input/output ports in some design, as shown below. Now, the user wants to inject faults in the first three positions.

$$FD \quad fd\_instance(clk, Din, rst, Q, Qn);$$

So, the user needs to define in the “user-defined-netlist.csv” file as follows:

$$FD; [1, 2, 3]$$

The RASP-FIT tool reads the file and adds the *FD* keyword in the file where all primitives are defined, and their positions are concatenated in the library containing positions. When this line of code parsed under RASP-FIT for bit-flip fault model, the output is as follows,

$$FD \quad fd\_inst(f0 \ \hat{} \ clk, f1 \ \hat{} \ Din, f2 \ \hat{} \ rst, Q, Qn);$$

In the above example, f0, f1, f2 represents the bit-flip faults in this line of the code.

The smaller design is considered to illustrate the example for the user-defined primitives or functions. This design is taken from the ISCAS’89 benchmark circuits named ‘s27.v’. In this design, the user has defined three D-flip-flops as ‘dff’ as a user-defined-primitive. This user-defined keyword (dff) must be added to the ‘user_defined_primitives.csv’ file with the desired locations for injection of faults as (dff; [2 3]). The user wants to generate three faulty copies of the design. Therefore, the user needs to provide this file as an input, select fault model (bit-flip in this case) and the number of copies (in this case 3) to the RASP-FIT tool. Fig. 7 shows the original Verilog design with the first faulty copy of the design. It is seen that ‘dff’ contains only two faults at the positions mentioned above.

After adding the user-defined-netlist file, the RASP-FIT tool can be used for the fault injection modification, fault injection testing, hardness analysis and compaction of test vectors. There are various benchmark designs written in Verilog HDL are considered for these functions. These benchmark designs are ISCAS’85, EPFL designs and some behavioural designs.

V. Conclusion

In this paper, a technical perspective and guidance for the use of this novel tool (RASP-FIT) are presented. It includes the generation of standalone applications in Matlab, installing the RASP-FIT tool on any host computer and usage of the tool for different functions. The RASP-FIT tool can modify the Verilog HDL code for various abstraction levels for fault injection analysis. Also, the tool helps designers and test engineers to perform testing, compaction and hardness analysis. All these functions are used for these fault models (e.g. bit-flip & stuck-at 1/0). The tool is fast, automatic, technology-independent and user-friendly.

```
// s27
// Original design
module s27(GND,VDD,CK,G0,G1,G17,G2,G3);
input GND,VDD,CK,G0,G1,G2,G3;
output G17;

wire G5,G10,G6,G11,G7,G13,G14,G8,G15,G12,
    G16,G9;

dff DFF_0 (CK,G5,G10);
dff DFF_1 (CK,G6,G11);
dff DFF_2 (CK,G7,G13);
not NOT_0 (G14,G0);
not NOT_1 (G17,G11);
and AND2_0 (G8,G14,G6);
or OR2_0 (G15,G12,G8);
or OR2_1 (G16,G3,G8);
nand NAND2_0 (G9,G16,G15);
nor NOR2_0 (G10,G14,G11);
nor NOR2_1 (G11,G5,G9);
nor NOR2_2 (G12,G1,G7);
nor NOR2_3 (G13,G2,G12);
endmodule

// s27
// Faulty Module 1
module s27_1(select,GND,VDD,CK,G0,G1,
    G17_f1,G2,G3);
input GND,VDD,CK,G0,G1,G2,G3;
output G17_f1;
wire G5,G10,G6,G11,G7,G13,G14,G8,G15,G12,
    G16,G9;
input [2:0] select;
wire fis=1;
reg f0,f1,f2,f3,f4,f5,f6,f7;
always @ (select) begin
  if (select == 3'd0) begin
    f0=fis;f1=0;f2=0;f3=0;f4=0;f5=0;f6=0;
        f7=0;end
  else if (select == 3'd1) begin
    f0=0;f1=fis;f2=0;f3=0;f4=0;f5=0;f6=0;
        f7=0;end
  .
  .
  .
  else begin
    f0=0;f1=0;f2=0;f3=0;f4=0;f5=0;f6=0;f7
        =0;end
  end
dff DFF_0 (CK,f0 ^G5,f1 ^G10);
dff DFF_1 (CK,f2 ^G6,f3 ^G11);
dff DFF_2 (CK,f4 ^G7,f5 ^G13);
not NOT_0 (G14,f6 ^G0);
not NOT_1 (G17_f1,f7 ^G11);
and AND2_0 (G8,G14,G6);
or OR2_0 (G15,G12,G8);
or OR2_1 (G16,G3,G8);
nand NAND2_0 (G9,G16,G15);
nor NOR2_0 (G10,G14,G11);
nor NOR2_1 (G11,G5,G9);
nor NOR2_2 (G12,G1,G7);
nor NOR2_3 (G13,G2,G12);
endmodule
```

Fig. 7. Code snippet (original design and modified design).

REFERENCES

[1] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *2014 9th IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, (Santorini), pp. 1–6, IEEE, May 2014.

[2] A. Benso and P. Prinetto, *Fault Injection Techniques And Tools For Embedded Systems Reliability Evaluation.* Kluwer Academic Publishers, 2003.

[3] J. Barboza, *Dependability Evaluation of a Critical System by means of Fault Injection Mechanisms.* PhD thesis, 2017.

[4] M. Desogus, L. Sterpone, and D. M. Codinachs, "Validation of a tool for estimating the effects of soft-errors on modern SRAM-based FPGAs," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 111–115, IEEE, Jul 2014.

[5] Z. Navabi, *Digital System Test and Testable Design Using HDL Models and Architectures*. Worcester, MA USA: Springer New York Dordrecht Heidelberg London, 2010.

[6] D. Kammler, J. Guan, G. Ascheid, R. Leupers, and H. Meyr, "A Fast and Flexible Platform for Fault Injection and Evaluation in Verilog-Based Simulations," in *2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*, pp. 309–314, IEEE, Jul 2009.

[7] V. Sieh, O. Tschache, and F. Balbach, "VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions," in *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, pp. 32–36, IEEE Comput. Soc, 1997.

[8] H. Ziade, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," *The International Arab Journal of Information Technology*, vol. 1, no. 2, pp. 171–186, 2004.

[9] W. Chao, F. Zhongchuan, C. Hongsong, and C. Gang, "FSFI: A Full System Simulator-Based Fault Injection Tool," in *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pp. 326–329, IEEE, Oct 2011.

[10] A. Rohani and H. G. Kerkhoff, "Rapid transient fault insertion in large digital systems," *Microprocessors and Microsystems*, vol. 37, pp. 147–154, Mar 2013.

[11] W. Mansour, R. Velazco, R. Ayoubi, H. Ziade, and W. El Falou, "A method and an automated tool to perform SET fault-injection on HDL-based designs," in *2013 25th International Conference on Microelectronics (ICM)*, (Beirut), pp. 1–4, IEEE, Dec 2013.

[12] W. Mansour, M. A. Aguirre, H. Guzman-Miranda, J. Barrientos, and R. Velazco, "Two complementary approaches for studying the effects of SEUs on HDL-based designs," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 220–221, IEEE, Jul 2014.

[13] W. Mansour and R. Velazco, "An Automated SEU Fault-Injection Method and Tool for HDL-Based Designs," *IEEE Transactions on Nuclear Science*, vol. 60, pp. 2728–2733, Aug 2013.

[14] M. Shokrolah-Shirazi and S. G. Miremadi, "FPGA-Based Fault Injection into Synthesizable Verilog HDL Models," in *2008 Second International Conference on Secure System Integration and Reliability Improvement*, (Yokohama), pp. 143–149, IEEE, Jul 2008.

[15] W. Mansour and R. Velazco, "SEU Fault-Injection in VHDL-Based Processors: A Case Study," *Journal of Electronic Testing*, vol. 29, pp. 87–94, Feb 2013.

[16] B. Rahbaran, A. Steininger, and T. Handl, "Built-in Fault Injection in Hardware - The FIDYCO Example," in *Second IEEE International Workshop on Electronic Design, Test and Applications*, (Perth, WA, Australia), pp. 327–327, IEEE, 2004.

[17] M. Jeitler, M. Delvai, and S. Reichor, "FuSE - a hardware accelerated HDL fault injection tool," in *2009 5th Southern Conference on Programmable Logic (SPL)*, (Sao Carlos), pp. 89–94, IEEE, Apr 2009.

[18] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi, "SCFIT: A FPGA-based fault injection technique for SEU fault model," in *2012 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, (Dresden), pp. 586–589, IEEE, Mar 2012.

[19] L. Naviner, J.-F. Naviner, G. dos Santos, E. Marques, and N. Paiva, "FIFA: A fault-injection–fault-analysis-based tool for reliability assess-

ment at RTL level," *Microelectronics Reliability*, vol. 51, pp. 1459–1463, Sep 2011.

[20] C. Dunbar and K. Nepal, "Using Platform FPGAs for Fault Emulation and Test-set Generation to Detect Stuck-at Faults," *Journal of Computers*, vol. 6, pp. 2335–2344, Nov 2011.

[21] A. R. Khatri, A. Hayek, and J. Börscök, "Validation of the Proposed Fault Injection, Test and Hardness Analysis for Combinational Data-Flow Verilog HDL Designs Under the RASP-FIT Tool," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, (Athens, Greece), pp. 544–551, IEEE, Aug 2018.

[22] A. R. Khatri, A. Hayek, and J. Börcsök, "Validation of the Proposed Hardness Analysis Technique for FPGA Designs to Improve Reliability and Fault-Tolerance," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 12, pp. 1–8, 2018.

[23] A. R. Khatri, A. Hayek, and J. Börcsök, *Applied Reconfigurable Computing*, vol. 9625 of *Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2016.

[24] M. Victoria, O. M. Querin, C. Díaz, and P. Martí, "liteitd a matlab graphical user interface (gui) program for topology design of continuum structures," *Advances in Engineering Software*, vol. 100, pp. 126 – 147, 2016.

[25] A. R. Khatri, A. Hayek, and J. Börcsök, "RASP-FIT: A Fast and Automatic Fault Injection Tool for Code-Modification of FPGA Designs," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, pp. 30–40, 2018.

[26] A. R. Khatri, A. Hayek, and J. Börcsök, "Fault Injection and Test Approach for Behavioural Verilog Designs using the Proposed RASP-FIT Tool," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 4, pp. 57–63, 2019.

[27] A. R. Khatri, A. Hayek, and J. Börcsök, "ATPG method with a hybrid compaction technique for combinational digital systems," in *2016 SAI Computing Conference (SAI)*, (London, UK), pp. 924–930, IEEE, Jul 2016.