Fault Tolerance Properties of Systems Generated with the Use of High-Level Synthesis

Jakub Lojda, Jakub Podivinsky, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology,
Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
{ilojda, ipodivinsky, kotasek}@fit.vutbr.cz

Abstract

During the last decades, electronic systems became an important matter of controlling many critical processes. However, those critical processes often require increased reliability. This requirement places pressure on system developers to make systems reliable. Because of ever growing chip-level integration, capabilities of electronic systems are expanding, and, thus, leading to more complex system architectures, the number of man-hours needed to develop such systems is significantly increasing. Many people believe the solution is to move the development to a higher level of abstraction (e.g. an algorithm level) and use the so-called High-Level Synthesis (HLS) for this purpose. In this research, we aimed towards a decision, whether the usage of HLS impacts the resulting reliability properties of the system, and, thus, whether the HLS-generated system matches reliability properties of its corresponding VHDL-implemented version. We found out that, for the selected set of circuits, HLS performs better in terms of resource consumption, but, also, which we consider surprising, in terms of reliability. For the selected set, HLS achieved better reliability by 3.03 *percentage points in contrast to the classical approach utilizing a traditional Hardware Description Language (HDL). In these experiments, no redundancy was intentionally inserted into benchmarking circuits.*

1. Introduction

Many critical processes that are controlled by an electronic system require a certainly higher level of reliability. These critical processes are usually controlled by a computer or, in general, by a digital system. A failure of such system would have a potential to cause irreversible damage, introduce high economical losses or even endanger human health. In some cases, the system is not easily accessible for an eventual repair, such as a space probe computer. The reparation of such device, while on its mission, might many times exceed the costs of the effort put to such system to enhance its reliability.

In case of Field Programmable Gate Arrays (FPGAs), reliability is mainly tested by evaluating the overall percentage of the so-called *critical bits* of the bitstream. A bit of the bitstream that participates in a functionality of a design (i.e. is *used* in the design) is called an *essential bit* [20]. A critical bit is a bit of the bitstream that causes a discrepancy on the system's behavior. The discrepancy of the behavior might be observable *immediately* or in a future on the output pins of the system. Some methods for system testing, such as the one published in [6], eliminate the problem of the future misbehavior by making elements that hold the internal state of the system *observable*. Other methods, such as that described in our previous publication [8], use certain time period, each tested bit is given, to show the discrepancy through its output pins. When computing the percentage of critical bits, it is important to focus on whether the percentage reflects the number of essential bits, which has a higher information value, or whether the percentage is based on the FPGA partition area, which might not reflect the actual usage inside of this partition. One approach that could be used to lower the number of the critical bits is called Fault Tolerance (FT) [4]. The approach of FT adopts the fact the failure may happen, but FT aims to mask such failures to protect the output data from errors.

With increasing chip-level integration of today's modern systems, the design of such systems is becoming a great challenge. Designers are required to utilize

978-1-5386-5710-2/18/$31.00 ©2018 IEEE

the given chip-area as much as possible, which significantly increases man-hours required. For this reason, designers are searching for a design flow that would allow them to abstract low-level details, and, thus, reduce the complexity of the development. One of such design flow methods is High-Level Synthesis (HLS). HLS is able to process the input description in the form of an algorithm (e.g. written in the C or C++ language) and output a functionally equivalent circuit description. The resulting implementation is usually on the Register Transfer Level (RTL) and written in the form of a Hardware Description Language (HDL; e.g. the VHDL or Verilog language). HLS usually provides means to accelerate the synthesized HW realizations such as loop *unrolling* or loop *pipelining* [2]. HLS is also able to go through the state space of all the possible hardware realizations and provides ability to find the realization with the desired hardware parameters. We believe the combination of these two approaches (i.e. FT and HLS) solves both of the problems. That is why we focus our research on such combination. In this paper, we are investigating the influences of HLS on the reliability of the synthesized circuit in comparison to the traditional HDL flow. We believe that this comparison is important not only for developers that design safety-critical applications and want to move to a higher level of abstraction, but also for our research focusing on reliable design automation on an arbitrary abstraction level. For the purpose of evaluation, we use our FT estimation framework. For our tests, we use the publicly available benchmark circuits from the *ITC'99* set [1], which we reimplemented in the C++ language intentionally for the purpose of testing HLS. As an HLS tool, we use the Catapult C [3].

This paper is organized as follows. Related work is discussed in Section 2. Our evaluation framework is presented in Section 3. The proposed experimental setup is discussed in Section 4. The results of our experiments are summarized in Section 5. Section 6 concludes the paper and presents our future plans.

2. Related Work

In the paper [12], the authors present a comparison of HLS and HDL approaches. For this purpose, they use HDL and C++ implementations of the *grid synchronization algorithm*. However, the authors focus on the resource consumption and the overall usability. The paper [10] discusses comparison of multiplication and division operations for the usage on Galois Fields (GF). The comparison focuses mainly on the size, frequency and latency of the implemented design. The authors of [14] propose a reliability evaluation of HLS-generated systems with and without the pipelining optimization applied and also with and without the usage of DSP's in the design. Their work can serve as a guide to select the particular HLS configuration for a designer working with the HLS approach on safety-critical systems. In [13], the authors published also an experimental evaluation of various HLS-based designs and evaluated their susceptibility to failure using fault injection. The authors compared the results with a soft-core processor-based implementation executing the SW implementation directly. The authors of [7] present an in-depth comparison of various open-source HLS tools, however, their comparison does not mainly focus on the reliability point of view. The authors of [9] also provide a detailed comparison, while focusing on open-source tools mainly.

Many publications exist, that compare various parameters of HLS-generated designs to the designed synthesized with the traditional HDL flow. However, the focus on reliability in the comparison of HLS and HDL designs, such as in this paper, is not addressed yet.

3. Evaluation Framework

For the purpose of the benchmarking circuits evaluation we use our FT-ESTimation (FT-EST) framework, which we previously published in [5]. The framework originally serves as a part of FT design automation SW, however, for this paper, we utilize its ability to fully evaluate the design while profitting from its acceleration capabilities. The FT-EST framework is designed to make the evaluation process of Units Under Test (UUTs) fast as much as possible. It incorporates several techniques to accelerate the evaluation and save the time needed for the reconfiguration, such as:

- parallel testing of multiple UUT instances,
- stimuli generation and outputs comparison is performed on the FPGA to remove any bottlenecks,
- after each test, just the UUT instances are refreshed back to their original bitstreams.

The FT-EST framework is composed of the HW part, which wraps the UUT instances, and the SW part, which controls the evaluation process and saves the results obtained from the HW. A new realization of the HW part must be synthesized every time a new UUT is evaluated, however, the process of the synthesis is fully automated, as the whole FT-EST framework is originally targeted towards testing of automatically synthesized FT circuits. The framework utilizes a well known concept of fault injection followed by functional verification of UUTs. The concept of the evaluation is based on a cycle that tests UUTs with the defined set of stimuli. This cycle that tests all the stimuli we call the *test cycle*. A cycle that tests all the Single Event Upsets (SEUs) (i.e. cycles through all the possible bits of the bitstream or all the possible fault scenarios) is called the *SEU cycle*. It is obvious that the test cycle is an integral part of the SEU cycle. The architecture of the framework is shown in Figure 1.

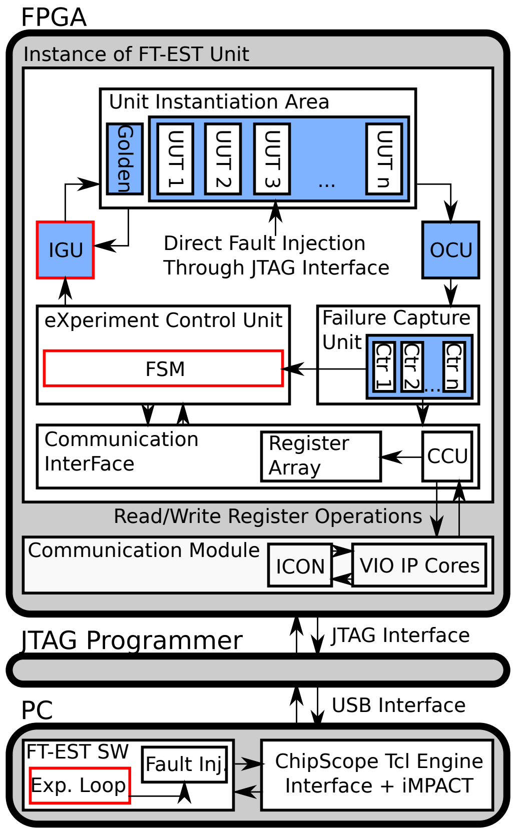

Figure 1. Simplified architecture of the FT-EST system; the parts highlighted in blue are dynamically and fully automatically generated, while the parts highlighted in red are to be provided by the designer to specify the experiment setup.

The HW part of the FT-EST framework is structured into various units, each of which has a particular function. The main component of the HW part is the **Unit Instantiation Area (UIA)**, which holds one *golden* instance of the UUT that provides reference results, and, therefore, is not subject to the fault injection. The UIA contains also the *ordinary* instances of UUTs to which faults are injected and impacts of these faults are evaluated. The input stimuli are generated inside of the **Input Generation Unit (IGU)**. Before testing of FT system of a particular nature, it is possible to tune the parameters of the IGU to suit the needs of the UUT. The UUT output values are compared against the golden unit inside of the **Output Compare Unit (OCU)**. The OCU then connects to the **Failure Capture Unit (FCU)**, which consists of counters that count the number of output mismatches per each UUT. The counter registers are addressable and readable from the SW part of the framework, allowing to analyze the number of erroneous outputs caused by a particular fault injection scenario (i.e. one particular bit or multiple bit-flips). The experiment flow of each *test cycle* is controlled by the **eXperiment Control Unit (XCU)**. The XCU, as well as the IGU, are dependent on the designer's choice. The autonomous execution of tests is controlled and supervised through the **Communication Interface (CIF)**. The CIF permits read and write operations to its configuration registers that trigger and control the evaluation process. In other words, this unit creates a technology independent communication interface. All of the components previously mentioned in this paragraph form the FT-EST unit, exposing its configuration capabilities through the CIF component. Besides this, another component called **Communication Module (CM)** opens the interface of the FT-EST unit through the implementation of the vendor-specific technology. In our case, the communication is performed through the Joint Test Action Group (JTAG) boundary scan interface in combination with the Universal Serial Bus (USB) programmer. As we are using Xilinx FPGAs, the technology behind the CM incorporates ChipScope Pro Integrated Controller (ICON) core [18] in combination with the Virtual Input/Output (VIO) cores [17].

The SW part of the framework utilizes our previously developed **fault injector** [11], which is able to filter the bits of the bitstream for a particular design block that are *essential* and utilized as Look-Up Table (LUT) contents at the same time. The SW also incorporates the SW counterpart of the CM, which is implemented using the **ChipScope Tcl Engine Interface** [16]. The reconfiguration to the initial state of the UUTs (i.e. reparation) is performed using the iMPACT tool. All of these tools are controlled using the **experimental loop**, which follows an experiment flow and also implements the *SEU cycle*.

4. Experimental Setup

We decided to investigate if and how HLS influences sensitivity to SEUs of circuits synthesized by it. This is useful not only for the designers using these tools but also for researchers utilizing these tools in their FT methods. We also decided to compare the results of HLS tools with the classical approach utilizing the VHDL language and conventional synthesis tool. For the evaluation purposes, we decided to choose the well-known benchmarking circuit set from the *ITC'99* [1]. We selected five particular circuits from this set. The selection is based primarily on the complexity of these circuits, which influences the evaluation time needed to fully test these circuits. The selected benchmarking circuits include:

1. **b01:** Finite State Machine (FSM) that compares serial flows,
2. **b02:** FSM that recognizes Binary Coded Decimal (BCD) numbers,
3. **b03:** resource arbiter,
4. **b04:** minimum and maximum computation,
5. **b05:** elaboration of the contents of a memory.

As all these benchmarking circuits come in the format of a Hardware Description Language (HDL), we used the original VHDL description to manually re-implement them in the C++ language, which allows us to process these circuits by HLS tool and compare the results with the traditional (i.e. HDL) synthesis flow. The overview of the original source code parameters and the reimplemented C++ code parameters, including the number of source code lines and variable instances, are shown in Table 1.

Table 1. Overview of the Original VHDL Codes and the Reimplemented C++ Source Codes.

Circuit Name	# In/Out Pins	VHDL Version Source		C/C++ Version Source		
		# Lines	# Processes	# Lines	# Variables	# Vars. Inside IF
b01	2/2	110	1	108	1	16
b02	1/1	70	1	72	1	3
b03	4/4	141	1	128	14	12
b04	11/8	102	1	92	13	8
b05	1/36	332	3	334	14	41

The resulting source codes were synthesized using the Catapult C HLS tools [3] to obtain the corresponding RTL VHDL implementations. Each VHDL implementation was then instantiated inside of the FT-EST framework to support the evaluation and synthesized using the Xilinx Integrated Synthesis Environment (ISE) 14.7 [15]. The evaluation was held on the ML506 FPGA board [19] utilizing Xilinx Virtex 5 FPGA technology. The flow of the experiments is summarized in Figure 2.

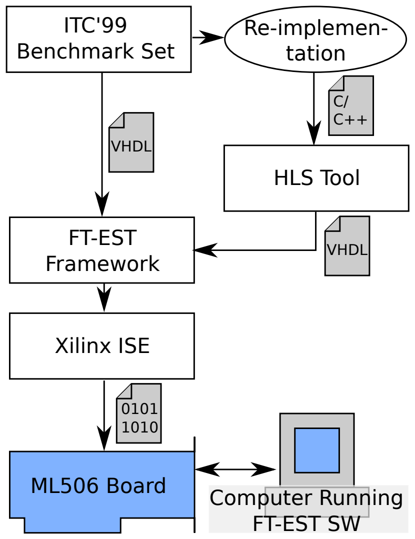

Figure 2. Overview of the experiments flow the aim of which is to find out the differences of FT properties between various circuits and HLS tools.

The FT-EST framework is set up to perform one verification (i.e. test cycle) per one utilized bit of the LUTs content bits of the bitstream. The input stimuli are generated inside of the IGU component, which utilizes a Linear Feedback Shift Register (LFSR) that is of 32 bits width length, although, each UUT's number of input pins is much smaller. UUT inputs are always connected to the first n bits of the LFSR, considering n equals the number of input pins of the UUT. The framework is configured to make exactly 400 million ticks of the clock signal before the counters inside of the FCU are sent to the computer, which saves them in a fault injection log file alongside with the address of the particular SEU injection bit. This way we obtain exact results of how many erroneous outputs out of the 400 million were generated for each injection scenario (i.e. flip of bit of the bitstream in this case).

5. Experimental Results

After the synthesis, we obtained 2 implementations per each of the selected benchmarking units. First, the resulting parameters considering resource utilization are shown in the first part of Table 2. We focused on LUTs, Flip-Flops (FFs) and MUltipleXers (MUXs). As can be seen, the results show that for the first four units b01 – b04, the resource consumption is at least equivalent, but, in some cases, the Catapult C-generated units are even smaller. Considering the VHDL to C++ transcription being straightforward, without any code optimizations, the results show, that the flow utilizing the Catapult C in combination with the Xilinx ISE results in more efficient implementations. For the b05 circuit, the number of utilized FFs is slightly more than doubled. As can be seen, the HLS flow did not utilize any MUX primitives in these benchmarks.

As the second, more important part of our research, we focused on the resulting reliability of the circuits. These results are, also, for better comparison, shown in the second part of Table 2. We monitored parameters of the LUT content bits having the essential function in the design (i.e. LUT bits that are utilized). We evaluated each of these essential bits using our FT-EST approach and obtained the number of critical LUT bits. The number of these critical bits serves as reliability indicator, although, it is important to relate this number to the essential bit utilization, which we did using the percentage in a next row of the table. The last row compares these results in the relation to the VHDL implementation. A number < 1.0 in this row indicates better reliability compared to the VHDL implementation and vice versa. As can be seen, for b01 and b03, the HLS reliability is worse, while for b02 and b04, the reliability is better. The b05, however, has *significantly* better reliability properties when synthesized using HLS. We were trying to find the reason, why some HLS-generated circuits achieved better reliability than their VHDL variant, and, except for the b05 circuit, we formed a hypothesis that this information *might* be in correlation with the number of variables inside of the C++ `if` statements. To save space, we added this information to Table 1. Generally, for this set of circuits, VHDL versions together contained 9.33 % of critical bits, while for HLS the reliability was better, counting 6.30 % of critical bits. For better clarity, the numbers of essential and critical bits are summarized in the chart shown in Figure 3. Critical bits in this case serve as an environment-independent unit to express the failure susceptibility of the design. The actual parameters, such as Mean Time To Failure (MTTF), can be calculated analytically or by a simulation later, after the actual failure rate of the particular circuit operation environment is determined.

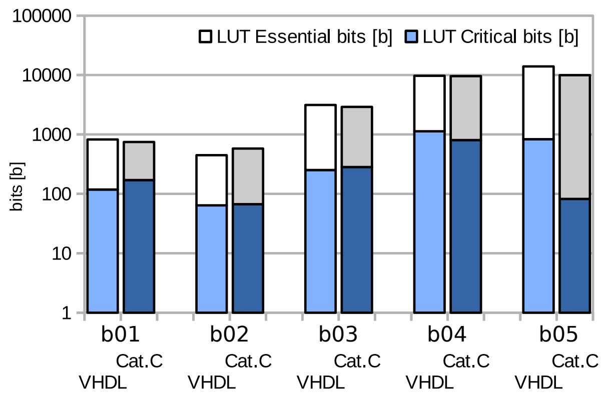

Figure 3. Chart comparing the levels of critical bits representations for the VHDL and HLS (i.e. Catapult C) flow; please note the Y-axis uses logarithmic scale.

6. Conclusions and Future Research

In this paper, our FT-EST framework was used to evaluate and compare the differences in reliability properties of systems generated using HLS tools and also using the traditional VHDL synthesis flow. In this evaluation, a subset of the well-known and publicly available benchmarking set, known under the name *ITC'99*, was used. This subset was reimplemented in the C/C++ language and synthesized using HLS flow. For the HLS synthesis, we used the Catapult C tools.

The results obtained in our experimentation show, that, for this set of circuits, in general, the reliability was better by 3.03 percentage points than the VHDL implementations. We consider this to be caused by the fact the HLS-generated circuit is processed twice (i.e. for the first time by the HLS tool and for the second time by the HW synthesis tool), and, thus, it has higher potential to internal optimizations that might result in a better performing circuit. Also, the consumption of resources is generally-speaking lower for the HLS set. We believe the HLS synthesis performs better during the optimization process. Nonetheless, the conclusion is that the realizations generated using HLS are almost equivalent to the VHDL described circuits, and, thus, a designer may expect similar behavior in terms of reliability, profiting from the higher level of abstraction.

As a part of our future work, we would like to evaluate our previously published approach to incorporate redundancy to HLS-generated systems and possibly tune the properties of the system to better suit multiple environments (i.e. various HLS tools). This work is part of a greater research focusing on FT system design automation. The actual information, whether the abstraction level of the circuit description affects its reliability, is very important prerequisite to be able to automate the design on various abstraction levels. It

Table 2. Comparison of Properties of Units Generated using Different Approaches.

	Circuit Name	b01		b02		b03		b04		b05	
	Variant	VHDL	Cat.C	VHDL	Cat.C	VHDL	Cat.C	VHDL	Cat.C	VHDL	Cat.C
Resources Utilized	LUTs # [-]	10	6	4	5	38	37	129	121	183	131
	FFs # [-]	10	6	4	5	35	35	67	67	36	85
	MUX # [-]	0	0	0	0	1	0	2	0	1	0
	Essential LUT bits [b]	704	576	384	512	2880	2624	8576	8768	13120	9856
	Critical LUT bits [b]	118	171	64	67	252	283	1130	805	831	82
	% of Critical LUT bits [%]	16.76	29.69	16.66	13.89	8.75	10.79	13.18	9.18	6.33	0.83
	% of Critical LUT bits Compared to VHDL [%]	1.0	1.79	1.0	0.83	1.0	1.23	1.0	0.70	1.0	0.13

should provide a little information to the whole puzzle and help to realize the idea of a source-description independent FT automation system.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II), the project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

[1] F. Corno, M. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *Design and Test of Computers, IEEE*, 17(3):44–53, Jul 2000.

[2] M. Fingeroff. *High-Level Synthesis Blue Book*. Xlibris Corporation, 2010.

[3] M. Graphics. Catapult HLS. `<https://www.mentor.com/hls-lp/catapult-high-level-synthesis/>`, 2017. Accessed: 2017-07-07.

[4] I. Koren and C. M. Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[5] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, and Z. Kotasek. FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation. In *Paper accepted for Presentation at Digital System Design (DSD), 2018 21st Euromicro Conference*. IEEE.

[6] C. López-Ongil, M. Garcia-Valderas, M. Portela-García, and L. Entrena. Autonomous Fault Emulation: A New FPGA-based Acceleration System for Hardness Evaluation. *Nuclear Science, IEEE Transactions on*, 54(1):252–261, 2007.

[7] W. Meeus, K. VanBeeck, T. Goedemé, J. Meel, and D. Stroobandt. An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, 16(3):31–51, Sep 2012.

[8] J. Podivinsky, O. Cekan, J. Lojda, M. Zachariasova, M. Krcma, and Z. Kotasek. Functional Verification based Platform for Evaluating Fault Tolerance Properties. *Microprocessors and Microsystems*, 52:145 – 159, 2017.

[9] S. Ravi and M. Joseph. Open source hls tools: A stepping stone for modern electronic cad. In *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–8, Dec 2016.

[10] A. Stanciu and C. Gerigan. Comparison between implementations efficiency of hls and hdl using operations over galois fields. In *2017 IEEE 23rd International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pages 171–174, Oct 2017.

[11] M. Straka, J. Kastil, and Z. Kotasek. SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems. In *14th EUROMICRO Conference on Digital System Design*, pages 223–230. IEEE Computer Society, 2011.

[12] F. M. Snchez, R. Mateos, E. J. Bueno, J. Mingo, and I. Sanz. Comparative of hls and hdl implementations of a grid synchronization algorithm. In *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, pages 2232–2237, Nov 2013.

[13] L. A. Tambara, J. Tonfat, A. Santos, F. L. Kastensmidt, N. H. Medina, N. Added, V. A. P. Aguiar, F. Aguirre, and M. A. G. Silveira. Analyzing reliability and performance trade-offs of hls-based designs in sram-based fpgas under soft errors. *IEEE Transactions on Nuclear Science*, 64(2):874–881, Feb 2017.

[14] J. Tonfat, L. Tambara, A. Santos, and F. Kastensmidt. Method to analyze the susceptibility of hls designs in sram-based fpgas under soft errors. In V. Bonato, C. Bouganis, and M. Gorgon, editors, *Applied Reconfigurable Computing*, pages 132–143, Cham, 2016. Springer International Publishing.

[15] Xilinx. ISE Design Suite. `<https://www.xilinx.com/products/design-tools/ise-design-suite.html>`, 2017. Accessed: 2017-07-07.

[16] Xilinx Inc. ChipScope Pro 11.4 Software and Cores User Guide. `<https://www.xilinx.com/support/`

documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf>, Dec. 2009. Accessed: 2018-02-15.

[17] Xilinx Inc. ChipScope Pro VIO Documentation. <https://www.xilinx.com/support/documentation/ip_documentation/chipscope_vio.pdf>, Sept. 2009. Accessed: 2018-02-15.

[18] Xilinx Inc. LogiCORE IP ChipScope Pro Integrated Controller (ICON) Documentation. <https://www.xilinx.com/support/documentation/ip_documentation/chipscope_icon/v1_05_a/chipscope_icon.pdf>, June 2011. Accessed: 2018-02-15.

[19] Xilinx Inc. ML505/ML506/ML507 Evaluation Platform User Guide. <https://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf>, May 2011. Accessed: 2018-05-29.

[20] Xilinx Inc. Soft Error Mitigation Using Prioritized Essential Bits. <https://www.xilinx.com/support/documentation/application_notes/xapp538-soft-error-mitigation-essential-bits.pdf>, Apr. 2012. Accessed: 2018-05-29.