Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS

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

***Abstract*—Due to the increasing demand for reliable computation in environments that require electronic systems to withstand an increased occurrence of faults (e.g. space, aerospace and medicine), new techniques of the so-called Fault Tolerance insertion arise. From another perspective, today's systems have become incredibly large and complex. Methodologies like High-Level Synthesis are used to reduce time to market and simplify the verification of the resulting system. In our research we focus on an implementation of Fault Tolerance into complex systems with the usage of High-Level Synthesis. In our approach, we are using newly designed Data Types that introduce redundancy on the functional level of an algorithm. In this student paper, our previously presented technique is extended by another means of redundancy and also by a new type of voting component. The systems incorporating various levels of redundancies using our approach are experimentally tested on the application of a robot controller. The paper also briefly presents the evaluation process and investigates its correct settings. The results show that the bit-based majority function is more suitable for usage with our Redundant Data Types.**

Keywords—High-Level Synthesis, Redundant Data Type, Level of Redundancy, Voter Component, CatapultC, Fault Tolerance.

I. Introduction

Some electronic systems require a very high level of reliability. The reason may be because the repair of these systems is very costly or in some cases even impracticable. Another reason for high reliability demand are systems whose failure would cause high economical losses or even could endanger human health. Two main approaches to high reliable systems construction exist. The first is the so-called *Fault Avoidance* (FA) [1], which is based on a strict selection of reliable components, thus increasing the overall system reliability. The second approach is the so-called *Fault Tolerance* (FT) [2]. The FT technique accepts that the system is composed of non-reliable components while trying to hide this fact and propagate the correct result in a prescribed time, even in the presence of faults. FT is based on an incrementation of redundancy, which can be spatial, temporal or information. Basically, a fault can be distinguished as *permanent* or *transient* (i.e. occurring only for a certain period of time). In our work, we focus on simulation of permanent faults mainly, as these faults have the potential to accumulate during the system operation.

As today's systems are becoming incredibly large and complex, methodologies such as *High-Level Synthesis* (HLS) are becoming popular. The movement to the higher layer of abstraction helps to reduce time to market and simplify the verification of the resulting system. HLS in this paper is understood as a collection of methods transforming a description in a higher-level programming language into its equivalent *Register Transfer Level* (RTL) representation. In this research, our goal is to bring the advantages of using unmodified HLS in the process of FT systems design. In this paper, the combination of the spatial and temporal redundancy is used to increase component reliability. The Catapult C [3] synthesis tool in collaboration with the Xilinx ISE tool [4] is utilized in this research. The Catapult C is set up with all optimizations off (i.e. no loop pipelining or unrolling is active), as the influences of those settings were, among others, studied in our paper [5].

Generally, two approaches incorporating FT into HLS can be distinguished: 1) HLS methods modifications; and 2) description source modifications. The method in [6] focuses on modified *data-paths* synthesis with concurrent error detection ability. The authors of [7] show a method of detecting *multi-cycle* transient faults. Another approach to error detection is presented in [8]. The authors of [9] present a heuristic algorithm for searching an optimal assignment of operations to *data-paths* while considering transient faults. In opposition to all the methods mentioned, the following approach moves the problem of reliability to a higher abstraction level (i.e. the function level). The authors of [10] developed a new data type that introduces the so-called *self-checking* (i.e. the error detection technique) into *data-paths* of HLS generated systems. The authors also consider the suitability of moving such problem to a higher level of abstraction in the context of the complexity of today's systems. In the paper [11], the authors evaluated this method on an application of the FIR filter. Generally, moving to a higher level of abstraction is important, as the level of chip-integration rises. In our research, we focus on developing a reliability insertion approach easily usable with today's modern HLS tools.

This paper is organized as follows. An overview of our FT method based on Redundant Data Types is proposed in Section II. Our experimental system setup and evaluation platform are presented in Section III. The experimental results are summarized in Section IV. Section V concludes the paper and suggests our plans for future research.

II. Redundant Data Types Method

Our method is based on the modification of the input source code before it is processed by HLS. Newly created Data Types (DTs) are used as a means of redundancy insertion. The redundancy is inserted to the source code by replacing the original DT name with the name of newly created so-called *Redundant Data Type* (RDT). RDT then incorporates redundancy to all the operators and storage elements associated with the corresponding variable instance. Each of RDTs represents one method of redundancy insertion, for example, the RDT *triple* represents the well known *Triple Modular Redundancy* (TMR); the RDT *quadruple* represents *Quadruple Modular Redundancy* (4MR); and the *quintuple* represents *Quintuple*

Modular Redundancy (5MR). Each RDT is connected to its so-called *original* data type, which implements the original data operations. The *original* data type is usually (but not limited to) one of the base types of the programming language used. In our research, RDTs are implemented using *C++ templates*.

In our previous work [12], we also used RDTs to evaluate importance of particular operation groups in a circuit. In our paper [5], three general types of *C++* modifications were identified: 1) variables (storage elements), 2) operators (arithmetic and logic operations) and 3) flow control statements. In this research, we focus on the first two types of modifications. The storage element multiplication is achieved by instantiating the variables of the *original* data type by the desired number of times. Then for unary operators, one RDT operation is performed on each instance of the *original* data type according to its original implementation. In order to allow automatic interconnections of subsystems using different reliability methods (i.e. binary operation of two non-equivalent RDTs, for example, the TMR and *duplex*), three cases must be distinguished for binary operations: a) *intra-data type* operations – RDT vs. RDT of equivalent redundancy types (e.g. TMR vs. TMR); b) *inter-data type* operations – RDT vs. RDT of different redundancy types – (e.g. TMR vs. *duplex*); and c) *original-data type* operations – RDT vs. its *original* (*unhardened*) DT (e.g. TMR vs. *unhardened* subsystem). These cases are schematically illustrated in Figure 1. For the ternary operator (i.e. the conditional operator) compatibility, the RDT must be able to cast its value to the *Boolean* data type. Each operation then includes an additional method that ensures self-synchronization (e.g. a majority function in the case of xMR).

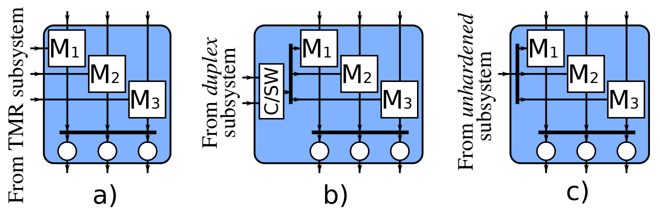

Figure 1: Three types of cases that can be distinguished when considering binary operations, (a) *intra-DT* operation between two TMR subsystems; (b) *inter-DT* operation between system with TMR and *duplex* hardening; and (c) *original-DT* operation between TMR and *unhardened* subsystems, [5].

As this approach works at the functional level, it allows for better compatibility between different HLS tools. It also benefits from the possibility of validation of implemented redundancy techniques before the development moves to the RTL. It is also easier to maintain the source code as the redundancy techniques are separated from the original code.

III. Experimental platform

The evaluation platform for testing FT properties presented in our previous work [13] is used for evaluation of the proposed methodology. Lots of real electronic systems are working together with some mechanical part. The mechanical part is usually controlled by its electronic controller. This is the reason why our evaluation platform monitors impact of faults not only on the electronic part, but also on the mechanical part. Our evaluation platform is based on the concept of the functional verification in combination with the artificial fault injection. A verified circuit (Design Under Test – DUT) is operating on *Field Programmable Gate Array* (FPGA), which allows us to inject faults directly into FPGA.

The architecture of the evaluation platform is shown in Figure 2. An electronic controller running on FPGA communicates with the simulation of a mechanical part which is running on a PC. The communication between FPGA and PC is accomplished through the Ethernet interface which is transformed to input signals for DUT. This transformation is done on the second FPGA. The verification environment which is also running on the computer monitors the communication between DUT and the mechanical part. The communication is compared with the reference model and a fault is reported in the case of a difference detection.

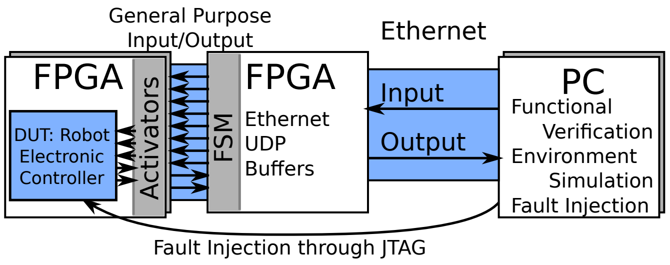
General Purpose Input/Output
Ethernet
FPGA
DUT: Robot Electronic Controller
Activators
FSM
FPGA
Ethernet
UDP
Buffers
Input
Output
PC
Functional Verification Environment
Simulation
Fault Injection
Fault Injection through JTAG

Figure 2: The architecture of our evaluation platform, [12].

The last tool working on PC is the fault injector. We use our previously published fault injector [14] which is able to inject permanent faults into a specified area of the FPGA. A permanent fault is simulated by flipping a configuration bit. Currently we are able to find a relation between bits of bitstream and the functional unit implemented on a specified place on the FPGA. For the experiments efficiency, it is very important to just inject faults into the utilized area of FPGA. Unfortunately, the current approach is only restricted on *Look-up Tables* (LUTs), so we just inject faults into the occupied LUTs. The fault injector is able to inject faults according to the specified strategy. It is possible to inject single faults during one run of the system or multiple faults within a various period. In this paper we use multiple injection with a specified rate.

The robot is exploring the maze in the Player/Stage simulation environment and its electronic controller is implemented into FPGA. The same experimental system is used in this paper, however, the robot controller is implemented with respect to the proposed methodology.

IV. Case Study and Experimental Results

For our experiments, the robot controller was fully implemented in the C++ language. Three redundancy methods were selected in combination with two majority function types: the TMR, 4MR and 5MR. For the purpose of these experiments, we decided to utilize the word-majority function (i.e selecting the word with the largest representation) and the bit-majority function (i.e for a particular bit position the value is selected by the majority representation). Generally, six RDTs (i.e. one RDT for each combination of redundancy and majority function type) were implemented according to the method previously mentioned in this paper. The overview of the RDTs used in our experiments can be seen in Table I.

TABLE I: The overview of the RDTs we have implemented as a part of our experimental work.

		Redundancy Method		
		TMR	4MR	5MR
Majority	Word	*triple*	*quadruple*	*quintuple*
	Bit	*triple_bit*	*quadruple_bit*	*quintuple_bit*

For each RDT, one robot controller unit was synthesized. Each robot controller unit was created by applying equivalent RDT to each of 30 variables in the controller's source code. Moreover, one reference robot controller unit without our approach applied (i.e. *noft*) was synthesized. The resource consumptions for each synthesized robot controller unit can be seen in Table II. As can be seen, the units with the *bit majority function* in their voter consume significantly less resources than the units with the *word majority*.

TABLE II: The overview of resource consumptions for each synthesized robot controller unit.

RDT Applied to the Robot Controller Unit Algorithm		Resource Consumption				
		Occupied Slices [-]	Slice Regs [-]	Slice LUTs [-]	Max. f [MHz]	LUTs bits [b]
	noft (no RDT)	338	708	634	249.5	19392
Word Majority	triple	611	999	1109	273.4	48704
	quadruple	647	1093	1479	239.2	73216
	quintuple	999	1560	2261	195.0	122880
Bit Majority	triple_bit	535	982	730	274.7	24480
	quadruple_bit	530	1102	755	309.3	26784
	quintuple_bit	596	1357	925	284.6	37632

A. *Fault Injection Intensity*

Redundancy in a circuit does not necessarily improve its resilience against faults. If we actually injected one permanent fault into our DUT per verification run, the results obtained would potentially be distorted by the occupied area of the circuit tested. The same situation occurs if we injected a constant number of faults per run. It is obvious that the injection ratio should reflect the size of the DUT. This corresponds to the fact that the failure manifestation probability is in direct proportion with the circuit area occupied. For this reason we suggest incorporating the size of circuit into the unit of fault injection rate. For this purpose we chose the unit of *bit* of the bitstream as the metrics for the circuit area occupancy. The resulting fault injection unit is $injection/s/bit$.

Moreover, in our research it is important to have an ability to compare results among various versions of experimental design implementations. For this purpose, we made a series of experiments to evaluate the impact of failure rate on the resulting *Mean Time To Failure* (MTTF). As the experiments' execution is very time consuming, we set the number of 500 verification runs for the *quintuple* RDT. The results for the failure rates of $4.5e$-6 to $0.5e$-6 can be seen in Figure 3. With a decreasing failure rate the number of failed runs also decreases while the MTTF increases slightly.

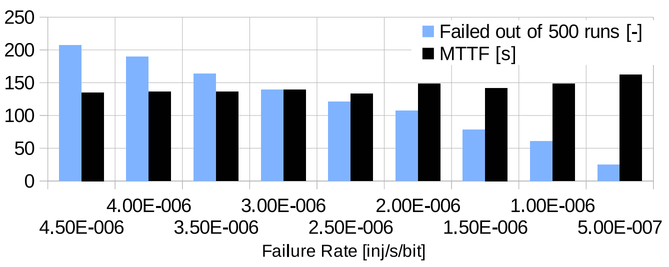

Figure 3: The impact of various failure rates on the number of failures and the resulting Mean Time To Failure (MTTF) for the controller utilizing *quintuple* RDTs.

As one maze exploration lasts for $204s$, we decided to choose the failure rate of $2.0e$-6 inj/s/bit. In this particular case, the MTTF of $148s$ was achieved, which is rational considering it is the 5MR implementation. As can be seen, while the failure rate is reduced, the number of failed runs decreases thus resulting in a less precise computation of the MTTF metrics because each run of the robot was limited to $324s$ (i.e. $204s$ + $120s$) in order to speed up the process of controller circuit evaluation (e.g. in case the robot is looped). It is important to note that the failure rate was selected only to make a fair comparison (according to the resolution scale provided by our evaluation platform) rather than to precisely simulate a real failure rate of any environment (e.g. the space).

B. *Effect of Redundancy Level and Voter Type*

We decided to examine $LUTs\_bits_{unit} \times 0.1$ number of runs per one robot controller unit as we assume that the number of verification runs should also be in direct proportion with the size of the DUT. The injection strategy was to inject faults into LUTs bits with the chosen fault rate of $2e$-6 inj/bit/s. Each fault injection bit was selected *uniformly at random* from all the utilized LUTs content bits. The parameters of the testing with the exact results obtained are shown in Table III.

TABLE III: The overview of the parameters of testing and the results obtained.

RDT Applied to the Robot Controller Unit Algorithm		Parameters of Testing			Results Obtained	
		LUTs bits [b]	Num. of Runs [-]	Fault Rate [inj/s/bit]	Failed Runs[%]	MTTF [s]
	noft (no RDT)	19392	1940	2e-6	21.24	131.05
Word Majority	triple	48704	4871	2e-6	18.99	139.40
	quadruple	73216	7322	2e-6	20.88	138.14
	quintuple	122880	12288	2e-6	21.34	141.06
Bit Majority	triple_bit	24480	2448	2e-6	18.91	128.68
	quadruple_bit	26784	2679	2e-6	20.87	132.88
	quintuple_bit	37632	3764	2e-6	25.05	130.08

For a better illustration, the results obtained are also shown in the chart in Figure 4. First, it is important to note that, the larger the circuit is, the more faults per second are injected, because the *fault rate* is related to the number of bits of bitstream, and, thus, the comparison is more fair. However, the difference between the *noft* and reliable units looks smaller from this perspective. As can be seen the results indicate that for the *triple*, the number of failed runs is lower which is the expected situation. For the *quadruple*, the percentage of failed runs is lower than for the unhardened unit, but still higher than for the *triple* unit. We assume this phenomenon is caused by the fact the 4MR redundancy occupies more area while the majority function still has to obtain three correct results (bits) out of four to select the correct output. The MTTF confirms this assumption as for the *triple* and *quadruple*, the MTTF is nearly equivalent. For the *quintuple* robot controller unit, the percentage of failed runs is slightly higher than for the *noft*. The MTTF suggests this is caused by a higher scatter of values and the fact the robot was tested exactly for the time period of one maze traversal plus 120 seconds. As can be seen the *triple_bit* controller unit achieved the failed runs ratio of $18.91\,\%$, which is the best result. In contrast, the MTTF decreased which would also suggest the scatter of values is higher in this case. For the *quadruple_bit*, the same phenomenon of increasing failed runs ratio can be observed. For the *quintuple_bit*, the percentage of failed runs increased and the MTTF decreased. In this case the bit-based majority function does not bring better results although the resulting circuit is still much smaller than its word majority version.

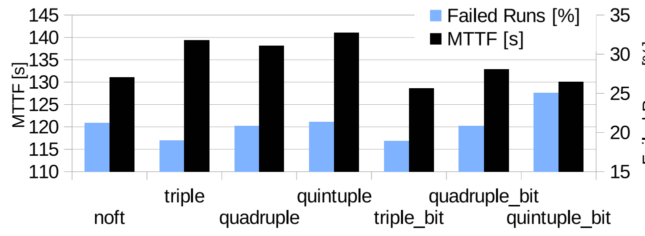

Figure 4: The overview of the results obtained through experimentation for each robot controller unit.

The conclusion of our experiments is that the bit majority function is more suitable for the purpose of usage with RDTs as it causes less overhead while keeping an almost equivalent reliability. In general, each RDT satisfies different requirements in terms of mission time or failure rate percentage.

C. Number of Verification Runs

As the number of verification runs was chosen purely empirically, we included a retrospective evaluation of the achieved precision. As a basic method of evaluation, we assume the higher the number of verification runs is, the more precise the results are. Therefore, we set the original results as a reference value and retrospectively calculated the results we would obtain if we set the number of verification runs lower. The reference value was then subtracted from each of the retrospectively calculated failure rates. By examining the deviation we are then able to get an idea of the achieved accuracy of our results. The differences in the failure rate ratios for each controller unit are shown in the chart in Figure 5. The number of verification runs for each controller unit is related to its number of LUTs bits (e.g. 0.1 in the chart represents $0.1 \times LUTs\_bits_{unit}$ verification runs).

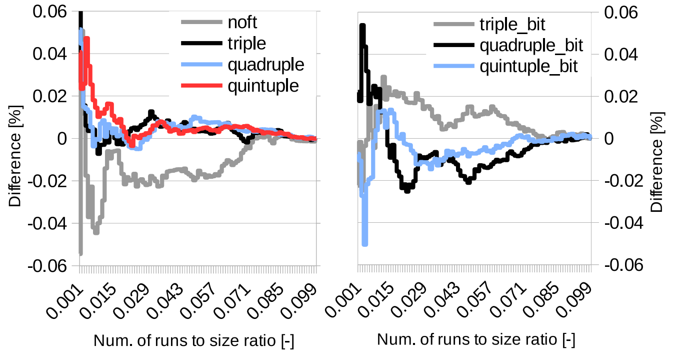

Figure 5: The retrospectively calculated failure rate differences for different numbers of verification runs.

As can be seen in Figure 5, starting from the ratio of 0.073 (i.e. $0.073 \times LUTs\_bits_{unit}$ runs) the difference from the result obtained at $0.1 \times LUTs\_bits_{unit}$ runs is kept under 0.01%. The important feature is that the differences stay at the same level after this point. Assuming the average converges to the ideal value, this suggests that the number of verification runs is sufficient considering the purpose of our evaluation.

V. Conclusion and Future Research

This paper briefly describes the approach of RDTs for usage with HLS and classifies this method in the context of other approaches. The main part of this paper evaluates the effect of the majority function type (i.e. word- and bit-based). The results show that the bit-based majority function leads to smaller circuits while keeping reasonable reliability. In addition, the paper also briefly explains the techniques utilized behind the process of the evaluation and selection of the evaluation parameters. The selected number of verification runs was retrospectively reviewed to verify its suitability.

As a part of our future research, we would like to find a key to select the proper redundancy method for a particular subsystem (i.e. function, expression etc.) and automate the process of this selection and source code modification.

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] J.-C. Geffroy and G. Motet, *Design of Dependable Computing Systems*. Kluwer Academic Publishers, 2002.

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

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

[4] Xilinx, "ISE Design Suite," https://www.xilinx.com/products/design-tools/ise-design-suite.html, 2017, accessed: 2017-07-07.

[5] J. Lojda, J. Podivinsky, Z. Kotasek, and M. Krcma, "Data types and operations modifications: A practical approach to fault tolerance in HLS," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 273–278.

[6] A. Antola, V. Piuri, and M. Sami, "High-level Synthesis of Data Paths with Concurrent Error Detection," in *Defect and Fault Tolerance in VLSI Systems, 1998. Proceedings., 1998 IEEE International Symposium on*, Nov 1998, pp. 292–300.

[7] A. Sengupta and D. Kachave, "Generating Multi-Cycle and Multiple Transient Fault Resilient Design During Physically Aware High Level Synthesis," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 75–80.

[8] K. A. Campbell, P. Vissa, D. Z. Pan, and D. Chen, "High-Level Synthesis of Error Detecting Cores Through Low-cost Modulo-3 Shadow Datapaths," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 161:1–161:6.

[9] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara, "High-Level Synthesis for Multi-Cycle Transient Fault Tolerant Datapaths," in *2011 IEEE 17th International On-Line Testing Symposium*, July 2011, pp. 13–18.

[10] C. Bolchini, F. Salice, D. Sciuto, and L. Pomante, "Reliable system specification for self-checking data-paths," in *Design, Automation and Test in Europe*, March 2005, pp. 1278–1283 Vol. 2.

[11] C. Bolchini, A. Miele, F. Salice, D. Sciuto, and L. Pomante, "Reliable system co-design: the FIR case study," in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings.*, Oct 2004, pp. 433–441.

[12] J. Lojda, J. Podivinsky, and Z. Kotasek, "Redundant data types and operations in HLS and their use for a robot controller unit fault tolerance evaluation," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 359–364.

[13] J. Podivinsky, O. Cekan, J. Lojda, and Z. Kotasek, "Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems," in *Digital System Design (DSD), 2016 19th Euromicro Conference on*. IEEE, 2016, pp. 487–494.

[14] 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*. IEEE Computer Society, 2011, pp. 223–230.