Delay analysis for FPGA devices operating at unconventional temperatures

1st S. Arash Sheikhoeslam
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
sarashs@ece.ubc.ca

2nd Parvez Chanawala
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
parvezc@ece.ubc.ca

3rd Pavithran Palanichamy
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
pavithran.palanichamy@gmail.com

4th André Ivanov
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
ivanov@ece.ubc.ca

***Abstract*—We developed an open-source programmable device characterization platform coded in Python. Our platform is effective at enabling the design of device reliability, performing accelerated aging, and measuring specific device timing/performance parameters. Our set of IPs is compatible with all Xilinx 7 series devices. We further used our platform to characterize the interconnect delays in a ZYNQ-7000 device. We measured the interconnect delays at various temperatures and established a relation between delay and temperature. We further looked into LUT primitives' degradation. Our results can complement timing analysis by the vendor tools for high-temperature applications such as server and vehicle applications.**

***Index Terms*—Reliability test, accelerated aging, FPGA, interconnect delay**

I. Introduction

IC and SoC reliability and aging have been the focus of our recent research. We have recently focused on Xilinx Zynq-7000 SoCs [1] and looked into how to test such parts to answer questions pertaining to their performance, reliability and aging. This has brought us to develop an open source embedded test platform that we refer to as *TYNQ*: Test platform on PYNQ.

We exploited these features to develop a user-friendly, generalizable embedded accelerated test platform. Figure 1 illustrates the principal elements of TYNQ and their functional relationships. The *IP Repo* element consists of a customizable set of test circuit building blocks (IP blocks). These IPs interact with the ARM core through the AXI protocol [2]. The specific interactions between these IPs and the ARM core(s) are enabled via a set of specially developed drivers also coded in Python. While we have focused our development and experiments on the Zynq-7000 SoC devices, TYNQ forms a platform to conduct experiments on any devices supported by the PYNQ framework.

The authors wish to thank Huawei for providing financial support for this work through the Huawei-UBC Joint Lab and the Natural Sciences and Engineering Council of Canada (NSERC) for providing support through its Alliance Program. We also wish to thank Su Yongjie from HiSilicon Technologies Company for his support and regular technical conversations about this work.

FPGAs have been at the heart of multiple studies on transistor aging and device reliability [3]–[9]. There are two main reasons for this particular interest. The first is the inherent interest in characterizing aging mechanisms and effects that affect FPGA performance and reliability over time. A second is the use of FPGAs as proxies to address the more general interest in the study and characterization of transistor aging and reliability in devices used in custom ICs and ASICs. In both scenarios, experiments based on platforms that use various external test and measurement instruments and data analysis algorithms tend to be difficult to reproduce precisely and accurately. Hence, a test platform that is fully embedded in the SoC device under test is an attractive alternative as it readily allows for the exact or quasi-exact reproduction of arbitrary experimental conditions. The TYNQ platform overcomes the above challenge.

We used TYNQ to analyze the impact of age-induced and temperature-induced delay in the ZYNQ-7000 FPGA devices. We measured the expected interconnect and LUT delay introduced due to operation at higher temperature as well as due to the CMOS degradation.

The remainder of this paper is organized as follows. In the next section, we describe TYNQ's hardware and software elements. In Section III we use TYNQ to perform our delay analysis. Section IV concludes the article. The Python codes, programmable logic bit-streams, and user manuals (wiki) can all be found in the following repository: https://github.com/sarashs/TYNQ

II. TYNQ Hardware and software

As per Figure 1, the core elements of TYNQ include a set of IPs, represented by the **IP Repo** block, along with a set of corresponding drivers, collectively referred to as **TestChip Driver**. The latter is represented in Figure 1 by the block

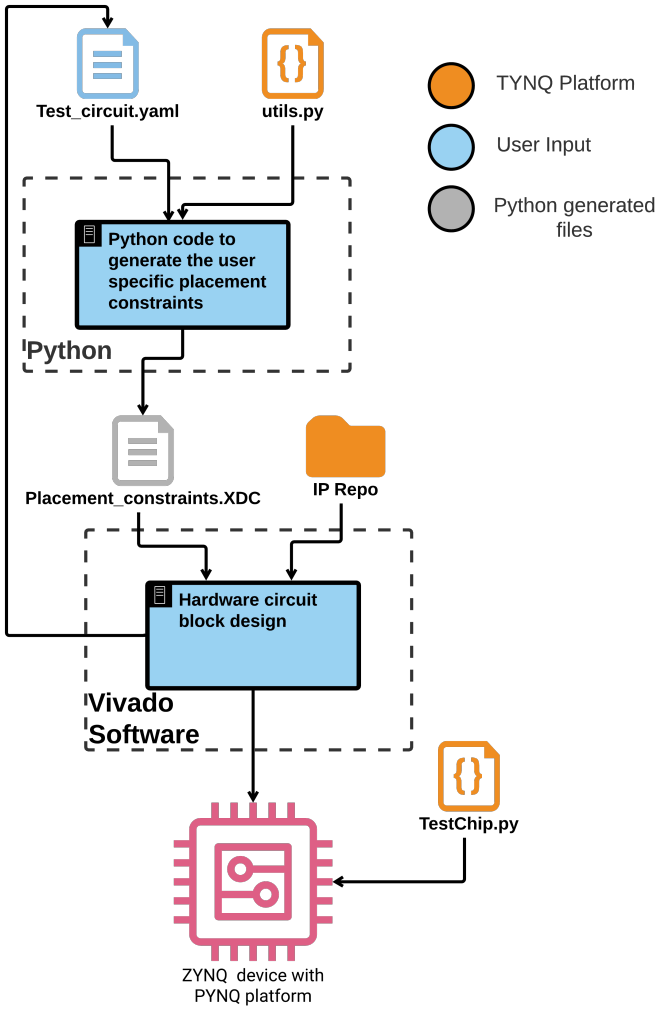


Fig. 1. Structural and functional illustration of TYNQ and its related elements. TYNQ itself consists of a set of IPs found in *IP Repo*, a Python utility module *utils.py* to generate placement constraints, and a driver module *TestChip.py* used to define the sensor structures and the required interactions with the sensors. The platform uses the Xilinx Vivado software platform to instantiate the specific test circuitry onto a Zynq SoC device.

TestChip.py. TYNQ also includes a set of "utility" Python codes to enable the instantiation of user defined experiments by generating user specific placement constraints for the IPs. These modules are represented by the **utils.py** block in Figure 1.

TYNQ's predefined test infrastructure hardware modules are customizable and designed to connect to the Zynq-7000 SoC processing system (ARM cores) via AXI-Lite interfaces. The current version of TYNQ includes five core test infrastructure modules: *Ring Oscillator (RO)*, *BTI Sensor*, *HCI Sensor*, *Self-Heating Element (SHE)* and *Temperature Sensor*.

Ring Oscillators (ROs) are commonly used as a basic test circuit structure for conducting experiments aimed at assessing aging and reliability. The frequency shift of an RO is commonly used as an indicator of a change in circuit environmental parameters (e.g., temperature) or for changes in

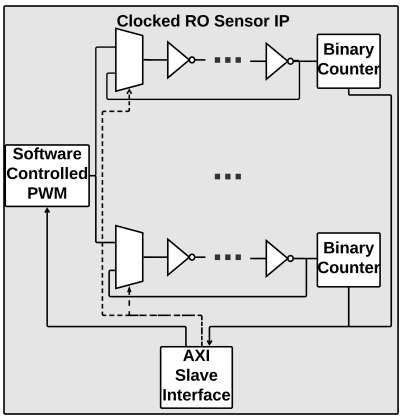


Fig. 2. Clocked RO circuit block diagram with a software-controlled PWM source as the clock.

device parameters (e.g., threshold voltage) typically associated with transistor aging. The RO frequency is determined by counting the number of rising edges of the RO's last stage occuring within a set period of time, 1 ms in our case (corresponding to 100,000 system clock cycles). We have also included **BTI and HCI sensors** for measuring Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) degradation. BTI is a dominant aging mechanism in current semiconductor technologies [8]. BTI degrades a transistor by increasing its threshold voltage. This increases circuit delays, thereby decreasing system performance, possibly to the point of failure, hence constituting a key reliability factor. Our BTI sensor circuitry is a slightly modified version of a circuit proposed in [7] which itself is a three stage ring oscillator with a NAND gate as the first stage. Each stage of the BTI sensors is implemented on one LUT-6 primitive and the entire sensor can be implemented using a single FPGA slice.

Our **clocked RO** sensor module is based on the circuit proposed in [5] and is illustrated in Figure 2. This sensor has the two modes of operation: *stress* and *measurement*. The mode is controlled by a single control signal fed to a 2-1 multiplexer at the input of each of the ROs. The control signal is set through software via the AXI-Lite interface (Figure 2 dashed lines). In line with the work of [5], when the sensor is set to *stress* mode, a PWM square wave signal drives the inverter chain. When in *measurement* mode, the inverter chain becomes an RO. While the original design by [5] uses a simple square wave, we introduced the PWM circuit to change the duty-cycle as well as the frequency of the input signal. The PWM signal frequency and duty-cycle are set by a software-controlled generator. Higher frequency stress is known to cause more HCI degradation and a lower frequency stress can be used as a baseline for measurement. Different duty-cycles can be used to control the expected BTI degradation.

Self-Heating Module: The Self-Heating Elements (SHEs) implemented in TYNQ's Self-Heating Module is based on Amouri's work [10], [11] where they proposed the use of controlled single-stage ROs as SHEs. TYNQ's Self-Heating

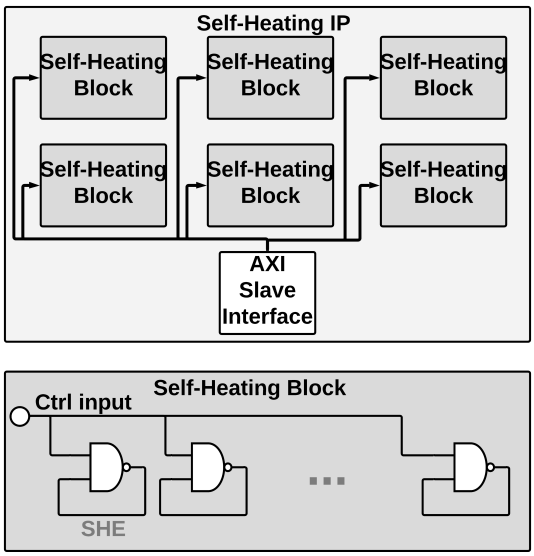
Module consists of a set of identical Self-Heating *blocks* controlled through a single input provided through the AXI slave interface. In turn, each self-heating block is comprised of a set of SHEs. Our self-heating module can have up to 64 such blocks, with each block comprising a maximum of 50 SHEs for a total of 3200 SHEs as illustrated in Figure 3. Note that the SHEs consume a lot of power and the maximum number of instantiated SHEs depends on the rest of the circuit.

The Zynq-7000 SoC has a built-in analog to digital converter (labelled *XADC*), which comprises a temperature sensor and on-chip power supply voltage sensors. TYNQ's Self-Heating Module uses the XADC's temperature reading as a feedback parameter to regulate the chip temperature (as further presented in a later section).

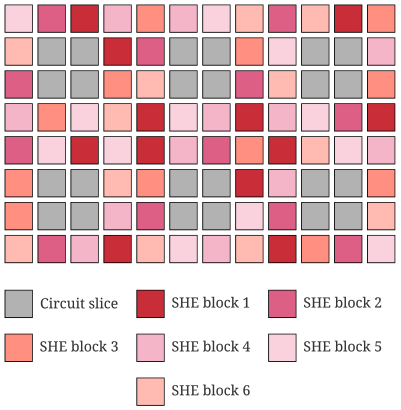
Our self-heating module is configured using two parameters Num_blocks and $Num\_SHE\_per\_block$ where total number of SHEs equals $Num\_blocks \times Num\_SHE\_per\_block$. The temperature is controlled through the driver software which implements a control loop running on the processing system by turning blocks of SHEs ON or OFF. The default control loop is as follows:

$i \leftarrow 0$
$SHE\_Blocks[0 : Num\_blocks - 1] \leftarrow OFF$ {Turn off all of the blocks.}
while $True$ **do**
  if $T_{current} < T_{desired} \pm T_{tolerance}$ **then**
    $SHE\_Blocks[i] \leftarrow ON$
    $i \leftarrow i + 1$
    if $i == Num\_blocks$ **then**
      $i \leftarrow Num\_blocks - 1$
    end if
  else if $T_{current} > T_{desired} \pm T_{tolerance}$ **then**
    $SHE\_Blocks[i] \leftarrow OFF$
    $i \leftarrow i - 1$
    if $i < 0$ **then**
      $i \leftarrow 0$
    end if
  else
    $Pass$
  end if
end while

where $T_{current}$ is the current temperature measured by the XADC, $T_{desired}$ is the desired operating temperature, and $T_{tolerance}$ is the acceptable temperature deviation range. A user can establish a stricter temperature control algorithm for greater stability and accuracy. For our experiments described later, this simple algorithm sufficed to generate acceptable temperature control. In order to uniformly distribute the heat through the area of interest, where the test circuits are located, the SHEs within each block are randomly scattered as per Figure 3 (b). Our Python utility tool allows the user to design the placement of test circuits (grey area in Figure 3 (b)) as well as the placement of the Self-Heating Module. The code *utils.py* assigns each SHE to a random Self-Heating block. This is to improve the temperature uniformity.



(a) Self-Heating Module circuit diagram showing SHE blocks with their respective control signal.



(b) Random placement of SHE blocks for a hypothetical case. The random placement of SHEs within a block uniformly distributes the heat and constant temperature in the region of interest.

Fig. 3. Self-Heating Module illustration of the placement of a total of 6 SHE blocks.

Temperature Sensor Module: The ability to implement multiple temperature sensors at arbitrary locations on an SoC die allows for better profiling of a die's temperature map and therefore allows for better control and characterization of temperature-dependent experiments. Using the frequency of ROs as a proxy for measuring local temperature has been proposed by several others, including [12], [13]. We too have implemented RO-based temperature sensors.

A. Software Modules

TYNQ is comprised of software modules to complement the hardware modules described above. These software modules are fully described in the repository for TYNQ. We just provide brief highlights below. **TestChip Driver (TestChip.py)**:

A Python driver class is provided which allows interactions between the processing system and the modules described above. A set of implemented functions are provided in the Appendix.

Python Utility Module (utils.py): We developed this module to allow users to generate constraints files for their specific test configurations. For instance, if there is a need for placing ROs or their stages in a certain pattern, our Python utility module can generate the constraints for various spatial configurations. Further details regarding the utility module as well as tutorials and examples are provided in the repository.

III. Interconnect Delay Characterization

We used TYNQ to analyze actual routing delays on a Zynq-7000 device. RO-based delay characterization of CMOS devices is found in the literature [14], [15]. TYNQ enables the ready analysis of actual post configuration delays associated with specific FPGA SoCs, a sort of *per-chip delay assessment*.

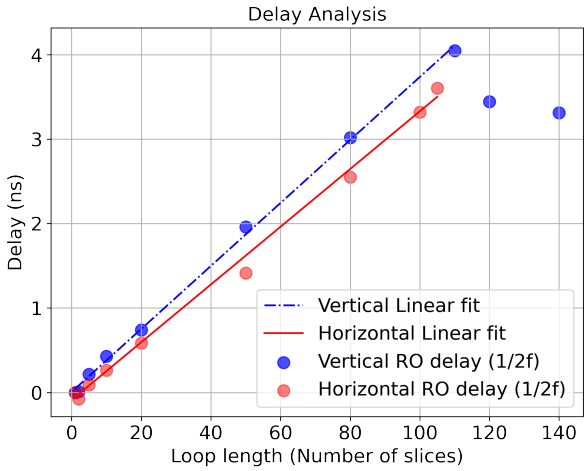
In our experiment, we used TYNQ's RO Module to configure 3-stage ROs followed by a delay (buffer) stage. We placed the 3-stage ROs in one slice and placed the delay stage in a varied number of slices away from the RO inverter stages as per Figure 4(a). This allowed us to effectively manipulate the length of the RO loop while keeping the number of stages fixed.

We analyzed the delays resulting from two sets of different configurations separated along the horizontal and routing layout axes of the SoC. Since the SoC interconnect resources are assumed to be according to a Manhattan grid style, these two sets of delay measurements are sufficient to provide a statistical estimate of the interconnect delays and their relation to the interconnect length (measured in units of number of slices.) Our first observation, as shown in Figure 4(b), indicates that the vertical resources' delay increases linearly until approximately 110 slices and then decreases as the separation distance increases to approximately 150 slices. As for the horizontal routing resources, the delay increases linearly with the length until approaching the maximum horizontal distance of 110 slices. Assuming an RC delay line (lumped or distributed), a wire delay is expected to increase quadratically with wire length [1]. Therefore, most of the delay is caused by routing switches and not the wires. The number of routing switches increases linearly with length (except for the lengths $>$ 110 slices). In such cases, the long vertical interconnects (vlong) are used more effectively which leads to fewer pass transistors and lower delays.

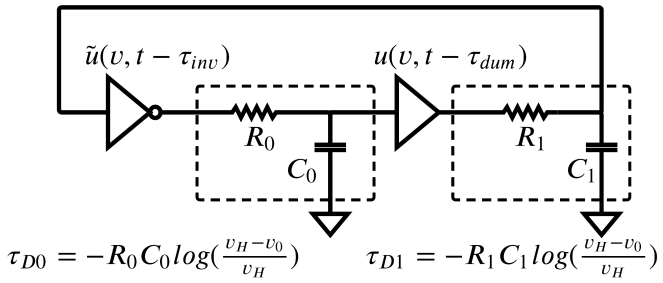
We further characterized the delays to find their temperature dependency. As depicted in Figure 4(c), we used a lumped RC model similar to [16] that we slightly modified to include our delay stage. We assumed an inverting step transition function for the inverters and a step transition function for the delay stage. Based on this model the total delay is the sum of delays in each segment of the circuit: $\tau_D = \tau_{inv} + \tau_{dum} + \tau_{D0} + \tau_{D1}$,

[1]For unit resistance and capacitance values r and c, the total RC value will be rcl^2 where l is the wire length.

(a) Simplified model of FPGA resources where 3 LUTs in one slice are used as the main stages for an RO and one LUT is used as a delay stage. The location of the delay stage determines the loop length (coloured loops). Two examples of horizontal and vertical loops are shown.



(b) The delay value in nanoseconds depicted for various loop lengths in the unit of slice number. Two types of loops (horizontal and vertical) were considered.



(c) Circuit diagram of a simplified model of our RO circuit with various sources of delay, namely the inverter delay τ_{inv}, delay stage's delay τ_{dum}, interconnect delays from and to the inverters (τ_0 and τ_1).

Fig. 4. TYNQ usage example: Statistical delay analysis and transmission line characterization of Zynq interconnects.

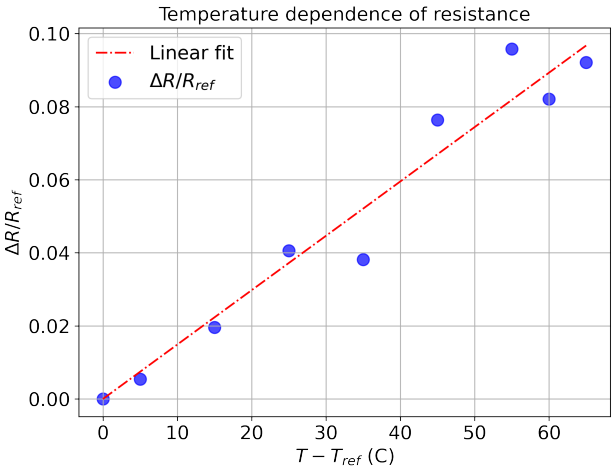


Fig. 5. The interconnect resistance changes as a function of the junction temperature. T_{ref} was set to 80°C.

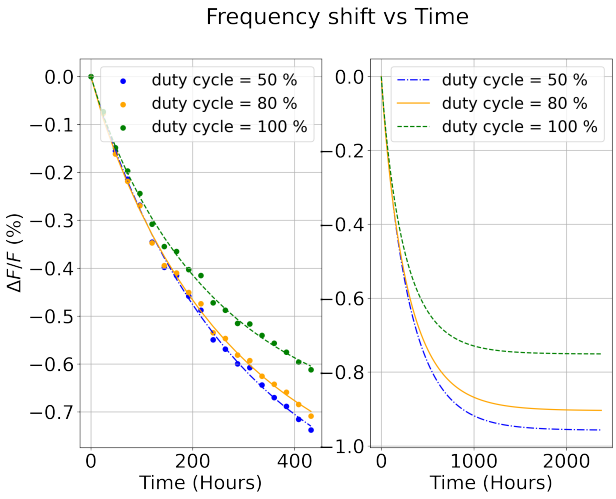


Fig. 6. Frequency shift as measured by our sensors along with decaying exponential fit (left) and extrapolation of exponential fit beyond 2000 hours (right).

with all these delay components being temperature-dependent. We performed the horizontal/vertical delay tests for various temperatures [75, 80, 90, 100, 110, 120, 130, 140] °C. Therefore, we expect the interconnect delay to be $\tau_{D0/1}(T) = -R_{0/1}(T)C_{0/1} \times log(\frac{v_H - v_0}{v_H})$ where v_0 is the input to the RC circuit and v_H is the threshold voltage for the logic High. We assumed that the capacitance is temperature independent. Considering the resistance temperature dependence as $R(T) = R_{ref}[1 + \alpha(T - T_{ref})]$, we have:

$$\tau_D(T)/\tau_D(T_{ref}) - 1 = \Delta R/R_{ref} = \alpha \times (T - T_{ref}) \quad (1)$$

where $\tau_D(T) = \tau_{D0}(T) + \tau_{D1}(T)$. Figure 5 shows the $\tau_D(T)/\tau_D(T_{ref})$ curve for various $T - T_{ref}$ values where the slope of the linear fit is the temperature dependence factor (α) which was computed to be 0.0015 from our experiments.

We have also analyzed the increase in the delay due to the degradation of CMOS transistors (mostly due to BTI and HCI). Note that other degradation mechanisms such as TDDB lead to device failure and are not considered in our delay degradation analysis.

We used the clocked RO sensors and stressed them at a temperature of 146° C. The PWM input enables us to study the effect of the duty cycle as well. Note that the Arrhenius nature of chemical reactions that lead to aging allows us to exponentially accelerate the aging process. This method of accelerated aging is common in HTOL tests. However, we are using a significantly higher temperature than what is common in HTOL tests.

We implemented three sets of 30 sensors with a stress PMW set to 100 Hz and varying duty cycles of 50%, 80% and 100% (i.e., DC). Each of these sensors was implemented using 3 inverter stages and data on RO frequency were collected for a total period of 400 hrs. The sensor data were collected every hour for a duration of 1 minute. The data was later averaged over 24 hours. Figure 6 shows the percentage of average frequency shift from the initial value. We used the data collected over 400 hrs to build a predictive model for degradation over longer periods. We assumed an exponential function fit to our experimental data as per the following [17]:

$$\frac{\Delta F}{F} = \alpha \times (1 - e^{-(\frac{t}{\beta})^{\gamma}}) \quad (2)$$

where t is time in hours, and α, β, and γ are the fit parameters. The resulting fitted functions are shown in Figure 6 (left-hand plot).

The data from our experiments show that the 50% duty cycle stress resulted in the highest amount of degradation, while the DC stress resulted in the least. We are not going to conjecture an explanation for this observation. Our goal was to find an empirical lower and upper bound on the LUT degradation over time which is given based on the asymptotic values in Figure 6 (right-hand side). In short Figure 6 and 5allow one to assess the interconnect and LUT primitive delays for extreme operating conditions (high operating temperatures when the device is aged).

IV. Conclusion

We presented the highlights of TYNQ, an embedded platform that we developed to characterize aging effects in PYNQ compatible Xilinx 7000 series devices. We developed TYNQ such that it offers considerable flexibility and capability for devising per-chip characterizations of timing performance. In particular, we developed TYNQ such that it be able to generate temperature stresses on the SoC device and thereby enable various accelerated aging tests and measurements. We reported on some of our own experiments with TYNQ here. We used TYNQ to perform a timing variation assessment on the routing resources of a Zynq-7000 SoC at different temperatures and computed the statistical interconnect delay (and resistance) temperature dependency. We also report on an experiment

where we perform a high-temperature accelerated test over a span of more than 400 hours. The experiment used no external heating, stressing and measurement equipment. We were able to estimate the HCI/BTI degradation effect on Zynq-7000 LUT primitives. Our results enable the designers to perform more accurate timing analysis for extreme operational conditions.

REFERENCES

[1] Xilinx, "Vivado design suite 7 series fpga and zynq-7000 soc libraries guide," 2021.

[2] S. Ramagond, S. Yellampalli, and C. Kanagasabapathi, "A review and analysis of communication logic between pl and ps in zynq ap soc," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. IEEE, 2017, pp. 946–951.

[3] A. Gupte, S. Vyas, and P. H. Jones, "A fault-aware toolchain approach for fpga fault tolerance," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 2, pp. 1–22, 2015.

[4] I. Stratakos, K. Maragos, G. Lentaris, D. Soudris, and K. Siozios, "Aging evaluation and mitigation techniques targeting fpga devices," Aristotle University of Thessaloniki, Tech. Rep., 2018.

[5] M. Naouss and F. Marc, "Design and implementation of a low cost test bench to assess the reliability of fpga," *Microelectronics Reliability*, vol. 55, no. 9-10, pp. 1341–1345, 2015.

[6] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. J. Irwin, and K. Sarpatwari, "Toward increasing fpga lifetime," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 2, pp. 115–127, 2008.

[7] A. Amouri, F. Bruguier, S. Kiamehr, P. Benoit, L. Torres, and M. Tahoori, "Aging effects in fpgas: An experimental analysis," in *2014 24th international conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–4.

[8] X. Guo and M. R. Stan, "Circuit techniques for bti and em accelerated and active recovery," in *Circadian Rhythms for Future Resilient Electronic Systems*. Springer, 2020, pp. 79–120.

[9] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung, "Degradation in fpgas: measurement and modelling," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 229–238.

[10] A. Amouri, J. Hepp, and M. Tahoori, "Built-in self-heating thermal testing of fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1546–1556, 2015.

[11] ——, "Self-heating thermal-aware testing of fpgas," in *2014 IEEE 32nd VLSI Test Symposium (VTS)*. IEEE, 2014, pp. 1–6.

[12] J. J. L. Franco, E. Boemo, E. Castillo, and L. Parrilla, "Ring oscillators as thermal sensors in fpgas: Experiments in low voltage," in *2010 VI Southern Programmable Logic Conference (SPL)*. IEEE, 2010, pp. 133–137.

[13] C.-A. Lefebvre, L. Rubio, and J. L. Montero, "Digital thermal sensor based on ring-oscillators in zynq soc technology," in *2016 22nd International Workshop on Thermal Investigations of ICs and Systems (THERMINIC)*. IEEE, 2016, pp. 276–278.

[14] B. P. Das, B. Amrutur, H. Jamadagni, N. Arvind, and V. Visvanathan, "Within-die gate delay variability measurement using reconfigurable ring oscillator," *IEEE Transactions on Semiconductor Manufacturing*, vol. 22, no. 2, pp. 256–267, 2009.

[15] M. Darvishi, Y. Audet, Y. Blaquière, C. Thibeault, and S. Pichette, "On the susceptibility of sram-based fpga routing network to delay changes induced by ionizing radiation," *IEEE Transactions on Nuclear Science*, vol. 66, no. 3, pp. 643–654, 2019.

[16] P. Z. Wieczorek and K. Gołofit, "True random number generator based on flip-flop resolve time instability boosted by random chaotic source," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1279–1292, 2017.

[17] X. Li, J. Qin, and J. B. Bernstein, "Compact modeling of mosfet wearout mechanisms for circuit-reliability simulation," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 1, pp. 98–121, 2008.

APPENDIX

TestChip.py: TestChip driver is child class of PYNQ's Overlay class with attributes and methods that allow the user to operate the IPs from within a Python script. It comes with the following methods at the time of writing this article (an up to date description can be found in our repository):

XADC_temp: measures the global temperature on Zynq-7000 device using the temperature sensor.

XADC_voltage: measures various voltages (programmable logic, processing system, BRAM etc) on Zynq-7000 device.

read_RO: measures the frequency of the ROs within the RO IP.

read_BTI: measures the frequency of the ROs within the BTI sensor IP.

read_HCI: measures the frequency of the ROs within the clocked RO sensor IP.

read_TMP: measures the local temperature using the RO-based temperature sensor IP.

HCI_set_pwm: sets the frequency and duty cycle values for the clocked RO sensor IP.

fix_temperature: sets the temperature to a given value.