Methodology to Forecast a BTI-Induced Accelerated Aging Test Result

by

Parvez Anwar Chanawala

B.Tech., Indian Institute of Technology Delhi, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

June 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Methodology to Forecast a BTI-Induced Accelerated Aging Test Result

submitted by Parvez Chanawala in partial fulfillment of the requirements for

the degree of Master of Applied Science

in Electrical and Computer Engineering

Examining Committee:

André Ivanov, Electrical and Computer Engineering, UBC
Supervisor

Prashant Nair, Electrical and Computer Engineering, UBC
Supervisory Committee Member

Steve Wilton, Electrical and Computer Engineering, UBC
Additional Examiner

Abstract

Integrated circuits are stressed at temperature and voltage levels beyond their nominal ratings for long durations (∼100s of hrs) during their development stage to study their reliability under nominal conditions. This is crucial to understand their operating lifetimes (typically in years) in their actual fields of use. The conventional HTOL test (an industry-standard reliability test to determine the intrinsic failure rate of ICs) is remarkably short as compared to the ICs' operating lifetime but still requires 1,000 hrs of elapsed test time. Future ICs' development may involve less time for such reliability tests due to the recent concerns being highlighted by most semiconductor manufacturers on reducing a product's time to market. To partially answer such concerns, I am introducing a methodology that models the results of such reliability tests.

To verify the feasibility of this method, several reliability experiments were conducted on Zynq-7000 FPGAs. To successfully perform those experiments, a reliability test platform was developed that can sustainably execute a high-temperature test for 1,000 hrs and requires minimum human intervention during the experiment. This platform is built on a commercial PYNQ-Z1 board that embeds the Zynq-7000 FPGA chip. To quantify the impact of thermal stress, several copies of a ring-oscillator-based test structure were implemented on the chip. Their free-running frequency was considered as a reference parameter to measure degradation.

I leverage an existing transistor-level aging model to develop a circuit-level aging model that can mathematically describe a circuit parameter's degradation as a function of time. This circuit-level aging model is then fitted onto the degradation data collected for a relatively shorter time frame to compute its parametric constants. Finally, with the known parametric constants, the model is used to determine how it fits the actual degradation data of the entire experiment.

An analysis reveals that the first ∼400 hrs of degradation data has sufficient information to forecast within a 3% accuracy margin the degree of degradation accomplished until the end of a 1,000-hour-long experiment. Subsequently, the analysis is applied to other test durations to study the effectiveness of this approach to other industry-standard reliability tests which are shorter than HTOL.

Lay Summary

To predict the reliability of a semiconductor device, it must pass through several industry-standard reliability tests where each test lasts for hundreds of hours. One such reliability test is HTOL (High-Temperature Operating Life) which is the longest, wherein the devices are stressed for 1,000 hrs at 125 ^{o}C to accelerate their aging process. This helps to determine the device's failure rate in its stipulated lifetime. However, due to such a long duration, the HTOL test at times negatively impacts a product's time to market. This can be prevented by predicting the test results in a significantly shorter time frame. This defined the primary goal of this research where an attempt is made to forecast the result of an accelerated aging test with a unique systematic approach. The feasibility of that approach was verified on Xilinx 28nm FPGAs by conducting several reliability experiments across temperatures for different test durations.

Preface

This thesis is the result of work carried out by myself, under the supervision of Dr. André Ivanov and the mentorship of Dr. S. A. Sheikholeslam.

Chapter 2 is based on the works primarily conducted by my research colleague, who at the time was a postdoctoral fellow, Dr. S. A. Sheikholeslam. I was responsible for testing and verifying the *Self-heating Module*, *Ring Oscillator (RO) Module*, and *TestChip Driver (TestChip.py)* of the TYNQ platform. I was the lead researcher for the experiments located in Chapters 4 and 5 where I was responsible for all significant areas of concept formation, data collection and analysis, and the manuscript composition.

A portion of Chapters 4 and 5 have been accepted for publication. P. Chanawala, I. Hill, S. A. Sheikholeslam, and A. Ivanov, "Prediction of thermally accelerated aging process at 28nm," in *2022 IEEE European Test Symposium (ETS)*, in press. I conducted all the experiments, data collection and analysis and composed most of the manuscript. Dr. S. A. Sheikholeslam developed the test platform to conduct the experiments, contributed to data analysis and manuscript edits. I. Hill contributed to data analysis and manuscript edits. Dr. A. Ivanov was the supervisory author on this project, providing guidance and feedback and contributing to manuscript edits.

Another portion of Chapters 4 and 5 is used to prepare a draft. P. Chanawala, I. Hill, S. A. Sheikholeslam, and A. Ivanov, "Forecasting Reliability Test Result from Partial Data for BTI-Induced Aging,". I conducted all the experiments, data collection and analysis and composed most of the draft. Dr. S. A. Sheikholeslam developed the test platform to conduct the experiments, contributed to data analysis and draft edits. I. Hill contributed to data analysis and draft edits. Dr. A. Ivanov was the supervisory author on this project, providing guidance and feedback and contributing to draft edits.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my deepest appreciation to my supervisor Dr. André Ivanov for providing his invaluable knowledge and expertise. I also could not have undertaken this journey without my defence committee, who generously provided their constructive feedback. Additionally, this endeavour would not have been possible without the generous support from the Huawei-UBC Joint Lab and NSERC, who financed my research.

I am also grateful to my research colleagues for their editing help, advice and suggestions. I would like to express my deepest gratitude to Dr. S. Arash Sheikholeslam for his regular technical support and guidance. I would also like to acknowledge the hardware support provided by Professor Guy Lemieux for my experiments.

Lastly, I would be remiss in not mentioning my family, especially my parents and my sister. Their belief in me has kept my inspiration and motivation high during this process. I would also like to thank my friends and community for all the entertainment and emotional support.

Dedication

To my parents and sister

Chapter 1

Introduction and Motivation

1.1 Reliability of Semiconductor Devices

The reliability of a semiconductor device can be understood as the duration (usually in years) for which a device will consistently perform to its expectations when operated under nominal conditions. Different semiconductor markets have diverse reliability requirements, as shown in Table 1.1. Many factors influence these requirements. The important component to be comprehended is how the semiconductor industry estimates the reliability of their semiconductor devices within a time-frame that is relatively much shorter than a device's expected operating lifetime. In simple terms, reliability is estimated by accelerating the aging process of a semiconductor device and subsequently analyzing its performance which is expected to be equivalent to that of a device that has been operated under nominal conditions for a duration equivalent to its required operating lifetime.

To determine and qualify the reliability of silicon ICs, the JEDEC (Joint Electron Device Engineering Council) defined several standard reliability tests for the semiconductor industry [3]. Often these standards are used as the basis to develop market-specific qualification tests.

Table 1.1: Reliability Requirements for Different Semiconductor Markets

Specification	**Consumer**	**Medical**	**Automotive**	**Aerospace and Military**
Operating Lifetime[*]	3-5 years	7-10 years	15 years	30 years

[*]The values in the table are a rough estimate based on various sources [4–9].

There are different aspects of reliability which are studied by each of the JEDEC's reliability tests. For example, in high temperature operating lifetime (HTOL) tests [10], the devices are stressed to determine their intrinsic failure rate during their expected operating lifetimes. In the early life failure rate (ELFR) tests [11], the stress conditions are designed to determine the failure rate during the initial phase of the ICs life. High temperature storage life (HTSL) tests [12] are performed to study the reliability of ICs that are not used immediately after production, but kept in storage for months for future use. Likewise, JEDEC defines various other reliability tests, and each of these have their own specific stress conditions, like die-temperature and voltage level, test duration, number of devices to be tested and pass/fail criteria [13].

1.2 Research Focus

An HTOL reliability test, whose stress conditions are described in JESD22-A108 [10], is particularly interesting. During an HTOL test, the devices are stressed for **1,000 hrs** at 125 ^{o}C and a supply voltage of $\sim$1.0-1.4$*$V$_{CC}$, because both temperature and voltage have been found capable of accelerating the aging process of a semiconductor device. This is remarkable at being one of the longest industry-standard reliability tests that consist of simultaneously testing 231 units (77 units x 3 lots) during which test none should fail, or else the entire test must be repeated with fresh lots. At times, this difficult requirement adds significant delay in a product's time to market. In the semiconductor industry, a product's time to market plays a crucial role in determining it's success. This implies that any such potential delay in a product's release can result in the loss of the company's respective market share, in turn impacting the company's revenue. Therefore, a reduction in this reliability test duration is of considerable importance.

Research is already underway to reduce test duration. One potential research avenue is the study of the *Acceleration Factor* (the factor by which the aging accelerates) during the HTOL test. S.W. Pae *et al.* in [14] and Robert Kwasnick *et al.* in [15], identified that the V$_{AF}$ (voltage acceleration factor) has significantly increased with advancements in the process node, thereby implying that the 1,000-hour HTOL is an over-estimation of the intended operating lifetime. This suggests reducing the HTOL test duration in accordance with the V$_{AF}$ of the respective technology. However, this approach is subject to fabrication technology.

Another potential approach to shorten duration is by predicting the HTOL test results. L. Yu *et al.* in [16] describe a method to select devices that could potentially "pass" such reliability tests, whereas W.T.K Chien *et al.* in [17] use WLR (Wafer Level Reliability) test results to predict HTOL test results. The latter approach is reasonable but still requires a correlation between the two reliability tests. Such works are primarily concerned with product-level V_{min} shifts or quick strategies for chip binning, whereas the research work described in this thesis, takes a more fundamental approach.

In this approach, an existing transistor-level aging model was leveraged to develop the circuit-level aging model which mathematically represents the degradation in a circuit parameter as a function of time. The value of the coefficient(s) of that circuit-level aging model are determined by merging the model's algebraic equation with the accelerated aging test data collected for a significantly shorter time frame. After obtaining the value of the coefficient(s), the model is used to generate a *predicted degradation curve* which can enable us to estimate the degradation in the remainder of the experiment. To measure the accuracy of prediction, the predicted degradation curve is compared to the actual degradation curve of the accelerated aging test.

To verify the feasibility of this approach, a significant amount of aging-test data was required. For that reason, several accelerated aging tests were conducted. To conduct those accelerated aging tests, we built a reliability test platform which could partially emulate the HTOL test conditions. The reliability test platform must provide a **stable** and **uninterrupted** operation, and maintain the desired **high temperature** for at least **1,000 hrs**. To achieve this, an **FPGA (Field-programmable Gate Array)** was chosen as a test platform because it is most readily available, has a relatively faster development time than ASICs for sensor-circuits and test setups, and has been successfully employed as a proxy in various reliability studies [18–23]. Most of such studies are either concerned with characterizing the influence of various aging phenomena on FPGAs, or with developing methodologies to improve FPGAs' operating lifetimes.

1.3 Reliability Physics of Semiconductor Devices

To better understand the approach, it is essential to understand the physics behind the aging of semiconductor devices, particularly as applied to the aging mechanisms that are of interest to this work.

1. **Bias Temperature Instability (BTI):** One of the most dominant aging mechanisms that occur in transistors, especially in current technology nodes, is BTI [24]. BTI-induced aging increases the threshold voltage (V_T) and reduces the mobility (μ) of a transistor under temperature and/or V_{GS} stress [25]. Consequently, BTI degrades a transistor's performance in terms of its current driving capability and switching speed, resulting in circuit slowdown.

2. **Hot Carrier Injection (HCI):** Like BTI, HCI is also characterized by the shift in the V_T and μ of a transistor, resulting in circuit slowdown. Although BTI and HCI have the same effect, charge carriers in the channel are responsible for HCI, since they gain sufficiently high velocity to get injected into the gate-dielectric [26].

3. **Time-Dependent Dielectric Breakdown (TDDB):** TDDB is characterized by the accumulation of oxide defects in the gate dielectric due to the application of the electric field (V_{GS}) in transistors. Unlike BTI and HCI, TDDB does not affect the V_T or μ of a transistor, but appears as the immediate failure of a transistor by effectively shorting the gate to the channel via the dielectric [4].

4. **Electromigration (EM):** EM is an aging mechanism that occurs in interconnects in metal wires. It is characterized by an increase in the interconnect resistance, caused by atomic drift in the direction of the electrons. Consequently, it increases the voltage drop across the interconnects, resulting in circuit slowdown. EM has an intense effect on the circuits that experience unidirectional current flow [27].

During an HTOL test, the accelerated aging observed in the devices is the cumulative result of all these aging mechanisms. The interaction amongst these mechanisms is complex and difficult to separate. However, if the test circuit and test conditions are chosen appropriately, we can enhance the effects of individual mechanisms. Therefore, as the first step toward this approach, the number of variables that impact the aging process were reduced so that the results can be conclusive. For that reason, our study's test circuit and conditions were set to enhance the effect of only one aging mechanism, chosen as being the **BTI**. Since **BTI** and **HCI** leads to circuit slowdown, their effects can be quantified by measuring the degradation in a circuit's parameter. Thus, our initial aim was to capture the effects of these mechanisms on circuits implemented on FPGAs. However, **HCI** was found to have negligible contribution at high temperatures. Therefore, the analyses were modified to focus on **BTI**.

To emulate the HTOL test conditions, we were required to simultaneously increase the temperature and voltage of the test chip. However, the supply voltage to the test chip, i.e. FPGA, was preset by the manufacturer using an external power regulator chip; both of which were embedded by default onto the PYNQ-Z1 board [28] (a commercial board that embeds the FPGA chosen for testing). Therefore, in our first step we could only increase the chip temperature to accelerate the aging process, thereby only partially emulating the HTOL test conditions.

1.4 Aim of the reliability experiments

This research mainly seeks to determine the feasibility of forecasting an accelerated aging test result from the data collected for a significantly shorter time frame. Further, our experiments were conducted at a wide range of temperatures (above nominal) to validate the consistency in the results across diverse temperature ranges.

Since forecast accuracy depends upon the quantity of data used for estimation, it was essential to analyze the discrepancy between the predicted and true values, of the accelerated aging test results, as a function of the time frame and sampling rate of the data collection. To study the benefit of this approach to other standard reliability tests conducted over a much shorter duration, including HAST and ELFR, the time-frame requirements for data collection as the duration of the experiment changed, to achieve a given level of prediction accuracy, was also analyzed.

1.5 Overview of the thesis structure

The thesis is organized into two parts. The first focuses on explaining the reliability test platform and its features. The second portion describes the reliability experiments conducted on the platform used and a detailed analysis of their results.

1.5.1 Part I

The first section covers the test platform and the flow of the experiments performed using it. Chapter 2 describes various features and modules of the reliability test platform. Chapter 3 discusses the importance of structuring the reliability experiments. It also explains the importance of automation in our experiments that made it possible to conduct the long-duration tests with minimal human intervention.

1.5.2 Part II

The second part covers the results obtained from the thermal stress experiments and various analyses of those results. Chapter 4 describes the setup of each of the thermal stress experiments conducted on the reliability test platform, and their measurement results. It also highlights the limitations of the test platform, indicating scope for improvement. Chapter 5 contains multiple levels of analysis that are performed on the measurement results to illustrate the feasibility of the approach. It also discusses about the benefit of the approach to other standard reliability tests of shorter duration than HTOL.

Following the analyses, Chapter 6 makes some conclusive remarks on this study, and presents new insights that arose during the process. It also describes a few limitations encountered during this approach and some final suggestions for future work that could make this approach more widespread than it currently is.

Chapter 2

Reliability Test Platform

2.1 FPGA

To perform all the thermal stress experiments that are described in this thesis the PYNQ-Z1 [28] board was employed to build the reliability test platform. PYNQ-Z1 embeds Xilinx's Zynq XC7Z020-1CLG400C chip which is a member of the Zynq-7000 SoC [29] family of FPGAs built on 28nm technology and is also the device under test (DUT). In the past, FPGAs have been widely used to study transistor and circuit-level aging and reliability for two primary reasons. The first is the motivation to increase the operating lifetime of FPGAs by characterizing the aging mechanisms and their effects that impact FPGA performance over the long run [19–23]. The second is to use the FPGAs as proxies to address the more general interest in the aging of transistors and the reliability of circuits used in custom ICs [18, 30].

In either scenarios, the test platform built to conduct those experiments rely on external lab equipment and measurement instruments, which makes their test setups difficult to replicate making results' reproducibility a challenge. To overcome this, the reliability test platform described here is embedded into the device under test, facilitating recreating the test setup.

2.2 TYNQ

The reliability test platform is TYNQ [1]: Test platform on PYNQ-Z1 board [28]. Figure 2.1 highlights the principal elements of TYNQ and their functional relationships. The *IP Repo* element is a repository containing a customizable set of programmable logic test circuits (IP blocks). The development of all the IPs took place in Xilinx *Vivado Design Suite - HLx Editions - 2020.1* software. These IPs when implemented on the FPGA chip, interact with the ARM cores of the FPGA through the AXI protocol [31]. The specific communication link between these IPs and the ARM cores is enabled via a set of specially developed drivers coded in Python. While the focus was on developing a test platform to conduct experiments on Zynq-7000 SoC devices, TYNQ is a platform that can be adopted to conduct similar experiments on any devices that support the PYNQ framework. The Python codes, programmable logic circuit's bit-streams, and user manuals (wiki) can all be found in the following repository: `https://github.com/sarashs/TYNQ`.

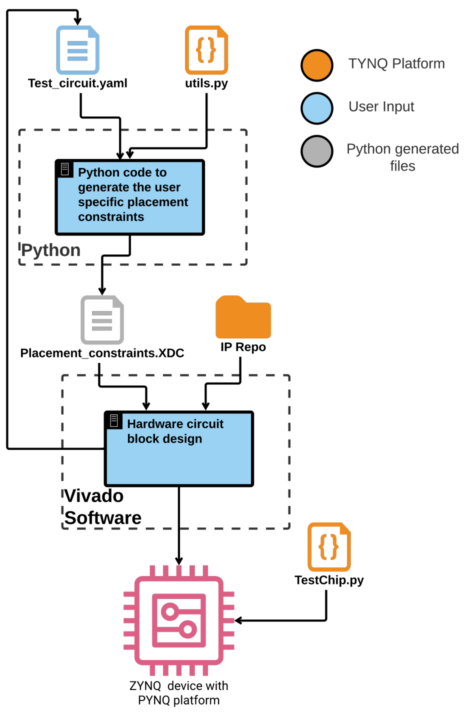

Test_circuit.yaml
utils.py
TYNQ Platform
User Input
Python generated files
Python code to generate the user specific placement constraints
Python
Placement_constraints.XDC
IP Repo
Hardware circuit block design
Vivado Software
TestChip.py
ZYNQ device with PYNQ platform

Figure 2.1: [1] A structural illustration of the TYNQ platform. TYNQ consists of a set of IPs stored in the *IP Repo*, a Python utility module *utils.py* to generate user-specific placement constraint files for IPs, and a driver module *TestChip.py* was used to enable user interaction with the IP circuits.

TYNQ's features are classified into two categories: hardware and software. Each class consists of several modules but the description here will be limited to the modules that were adopted to conduct the experiments described in this thesis.

2.2.1 TYNQ: Hardware

TYNQ's test infrastructure hardware modules (IP blocks) are customizable and designed to connect to the Zynq-7000 SoC processing system (ARM cores) via AXI-Lite interfaces as shown in Figure 2.2. The current version of TYNQ includes five core modules: *Ring Oscillator (RO)*, *BTI Sensor*, *HCI Sensor*, *Self-heating Module* and *Temperature Sensor Module* [1]. *BTI, HCI* and *Temperature Sensor Modules* are out of the scope of this work, and so will not be described here.

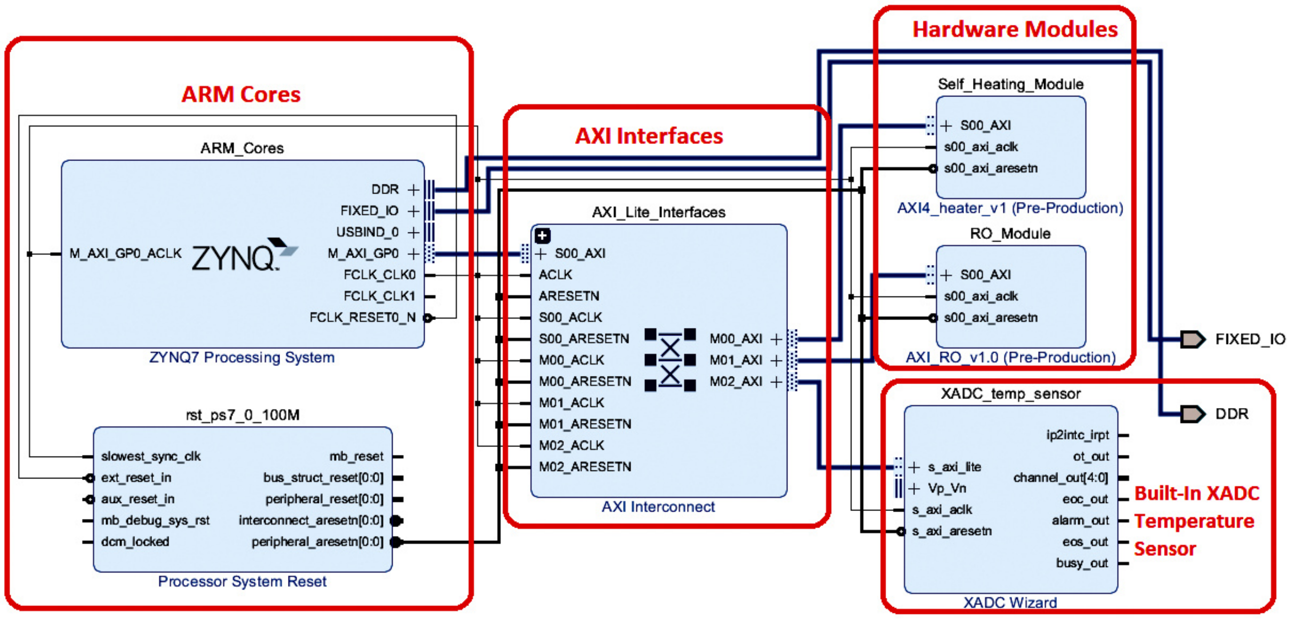


Figure 2.2: Vivado Block Design window: The connection of the IP blocks (**Hardware Modules**) with the Zynq-7000 SoC processing system (**ARM Cores**) via the AXI-Lite Interface (**AXI Interfaces**).

Ring Oscillator (RO) Module

Ring Oscillators have been widely used as basic test architectures to perform various kinds of accelerated aging and reliability experiments [18, 20, 21, 30]. A shift in their frequency is applied as a measure to capture change in the circuit's environmental conditions (including voltage and temperature), and/or to study drift in its transistors' parameters (such as μ and V_T), caused by aging. For the experiments discussed in this thesis, the TYNQ's RO architecture, illustrated in Figure 2.3, was modified because the given RO architecture lacks control over the oscillation frequency. The voltage signals between any two stages are not exactly square waves and neither possesses a precise 50% duty cycle. To have achievable control over the frequency of oscillation, sharp voltage transitions and steady voltage levels at all nodes, and to be able to guarantee that the stress on the oscillator is a quasi-ideal AC stress of 50% duty cycle, the architecture was modified to that portrayed in Figure 3.1. The modified circuit architecture is described in detail in the next chapter.

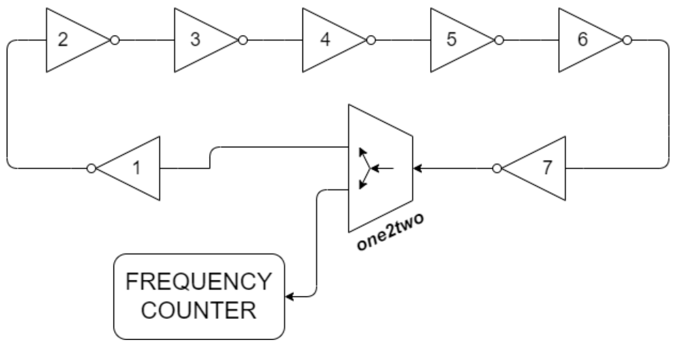
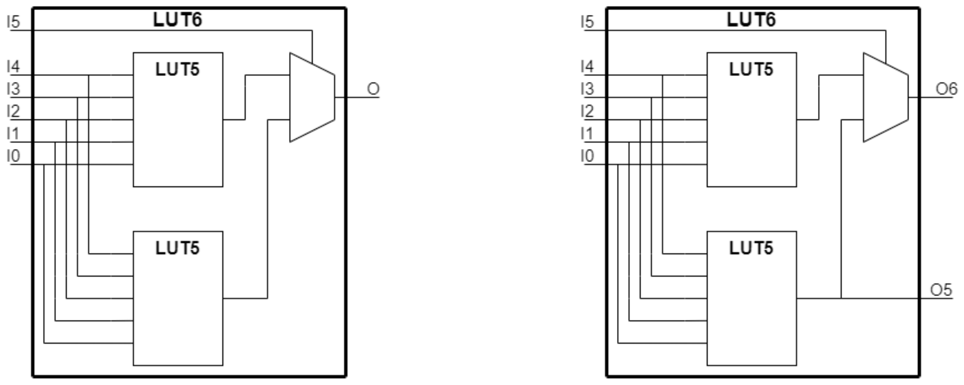


Figure 2.3: [1] TYNQ's RO Architecture.

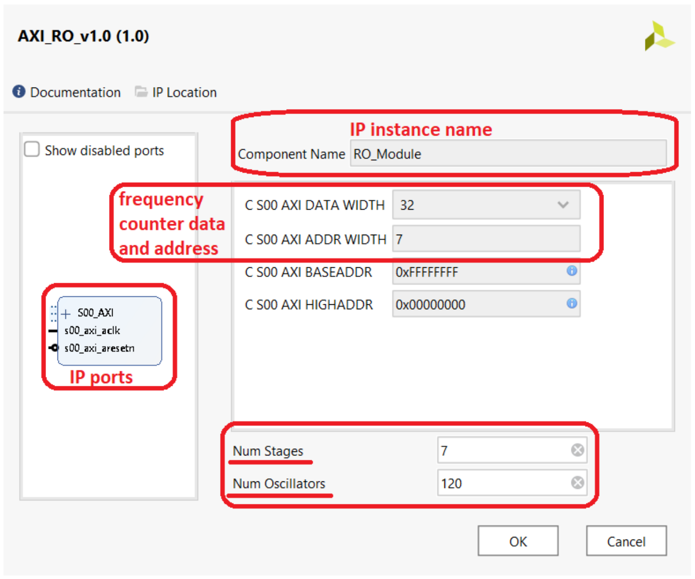


(a) *LUT6* (b) *LUT6_2*

Figure 2.4: [2] 6-Input Look-up Tables on Zynq-7000 SoC FPGAs with varied numbers of outputs.

Each inverter stage in the RO architecture is built upon a 6-input (*I0, I1,..., I5*) Look-up table *LUT6* with a single output *O*, shown in Figure 2.4a. Only *'I0'* is used for input whereas the remaining inputs are set to 0 and the SRAM bits (constituting the output values for each input combination) are pre-set to define the boolean function, *NOT*. Appendix A contains this inverter's Verilog instantiation code. Along with an odd number of inverter stages, the RO module possesses an additional stage, *one2two*, which does not invert its input but duplicates it to two different nets: one feeding input back to #1 inverter and the other to the *Frequency Counter*, to monitor the oscillation frequency of the RO. This stage was also constructed from a 6-input Look-up table, but has two outputs, *LUT6_2*, as shown in Figure 2.4b. Appendix B contains the Verilog instantiation code of the *one2two* stage.

The RO module has the flexibility to set the number of inverter stages in each ring oscillator, and the total number of such oscillators, to be implemented on the FPGA. These settings can be found during the instantiation of the RO IP under the IP customization window in Vivado software, as shown in Figure 2.5. The *frequency counter* data and address widths were predefined. However, a user can modify them by manipulating the source code.


AXI_RO_v1.0 (1.0)

Component Name RO_Module

C S00 AXI DATA WIDTH 32

C S00 AXI ADDR WIDTH 7

C S00 AXI BASEADDR 0xFFFFFFFF

C S00 AXI HIGHADDR 0x00000000

Num Stages 7

Num Oscillators 120

Figure 2.5: The Vivado IP Customization window: A **RO Module** customization of its number of stages (*Num Stages*) and the total number of ROs (*Num Oscillators*).

Self-heating Module

The purpose of this module is to raise the chip temperature to the desired levels to thermally stress the test circuits (like **Ring Oscillators**) that are implemented on the same chip, thereby removing dependency on the external temperature controllers known as thermal chambers (including ovens and thermostream) which are expensive and not necessarily available/accessible to all.

TYNQ's *Self-heating Module* [1] consists of numerous identical *SHE blocks* (self-heating blocks), in which each block is controlled with an input signal received from the processing system via the AXI slave interface, as shown in the Figure 2.6. Each block is composed of several identical Self-heating Elements (SHEs). Each SHE is a controlled single-staged RO, as proposed by A. Amouri *et al.* [32, 33]. The module has the flexibility to set the number of SHEs per block and the total number of such blocks, to be implemented on the FPGA. These settings can be found during the instantiation of the *Self-heating Module* under its IP customization window in the Vivado software, as shown in Figure 2.7. Note that the SHEs consume a significant degree of power and therefore, the maximum possible number of instantiated SHEs depends on the maximum power limit setting of the PYNQ board.

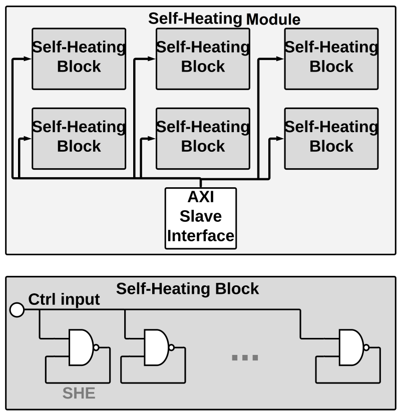
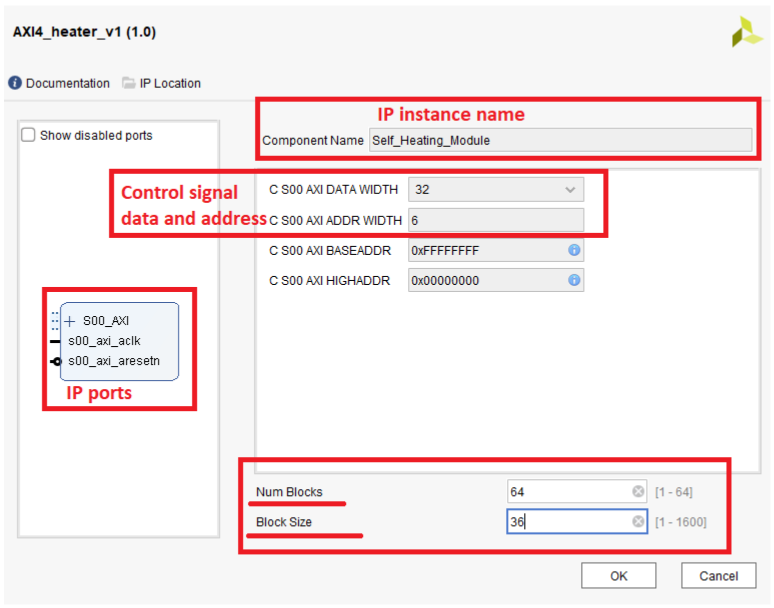


Figure 2.6: [1] A *Self-heating Module* circuit diagram displaying six SHE blocks with their respective control signals received from the AXI Slave Interface. Each SHE block consists of numerous SHEs.

The Zynq-7000 SoC FPGA has a built-in ADC (labelled *XADC*), which consists of a temperature and on-chip power supply voltage sensors. TYNQ's *Self-heating Module* utilizes the temperature sensor's reading as a feedback parameter to regulate the die temperature. The module is customizable and its configuration is based on two parameters Num_blocks (the total number of SHE blocks) and $Num\_SHE\_per\_block$ (the total number of SHEs per block), thereby setting the total number of SHEs to $Num\_blocks\ \times$



AXI4_heater_v1 (1.0)

Show disabled ports

Component Name: Self_Heating_Module

C S00 AXI DATA WIDTH: 32

C S00 AXI ADDR WIDTH: 6

C S00 AXI BASEADDR: 0xFFFFFFFF

C S00 AXI HIGHADDR: 0x00000000

S00_AXI
s00_axi_aclk
s00_axi_aresetn

Num Blocks: 64 [1 - 64]

Block Size: 36 [1 - 1600]

Figure 2.7: The Vivado IP Customization window: A **Self-heating Module** customization of the total number of blocks (*Num Blocks*) and number of heating elements per block (*Block Size*).

$Num\_SHE\_per\_block$. The temperature is regulated through the *TestChip Driver* (a TYNQ's software module) which implements a control loop running on the processing system to individually turn each of the SHE blocks ON or OFF by following a default temperature control algorithm [1], which 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 instantaneous temperature measured by the XADC, $T_{desired}$ is the desired chip temperature, and $T_{tolerance}$ is the acceptable temperature deviation range. The algorithm is customizable such that a user can implement a stricter control algorithm for greater precision. For the experiments described in the thesis, this simple algorithm provided acceptable ($\pm 1^o$C) temperature control.

2.2.2 TYNQ: Software

The hardware modules of TYNQ interact with the FPGA's processing system via a set of specifically developed drivers collectively referred to as the **TestChip Driver** [1], represented in Figure 2.1 by the block, **TestChip.py**. To aid users in placing the hardware module circuits in a desired configuration on the FPGA, the TYNQ platform provides a set of "utility" Python codes [1], represented by the **utils.py** block which generates user-specific placement constraint files for each of those hardware modules. These software modules are fully described in the TYNQ's repository. Their brief highlights are as follows.

TestChip Driver (TestChip.py)

This software establishes a communication link between the user and all the hardware modules implemented on the FPGA, via the PYNQ framework. Specifically, this script enables the user, during an experiment, to read the frequency measurement from the *Frequency Counters* of all the *RO Modules*, and the instantaneous on-chip temperature and voltage levels, and regulate the die temperature by controlling the *Self-heating Module*. For the reliability experiments later discussed in this thesis, this module was extended by adding certain features that aid in the automation of the experiment. These features will be discussed in the next chapter. The default set of implemented functions is provided in Appendix D.

Python Utility Module (utils.py)

The purpose of this module is to enhance users' ability to generate constraint (placement) files for the aforementioned hardware modules as per their desired configurations. For instance, if it is necessary to place ROs or their stages and/or thousands of SHEs in a particular pattern, the Python Utility module can generate their necessary constraint files. Further details regarding the utility module as well as tutorials and examples are provided in the repository.

2.3 Conclusion

This chapter briefly described various features of the reliability test platform: TYNQ, compatible with the Xilinx 7000 series FPGA devices that supports the PYNQ framework. Each of the modules described plays an important and unique role in all the experiments: the *Self-heating Module* helps to raise and maintain the die temperature whereas the *RO Module* acts as a test circuit to quantify the impact of thermal stress. The *Python Utility Module* made it feasible to generate placement commands for hundreds of ring oscillators and thousands of required SHEs whereas the **TestChip Driver** provided a means to centrally communicate with all the hardware modules. The next chapter describes the structure and flow of the long reliability experiments which were built upon the modules of the TYNQ platform.

Chapter 3

Reliability Experiment Structure

3.1 Necessity for Structuring the Reliability Experiments

Every semiconductor device undergoes some aging process throughout its lifetime, due to which its performance degrades. This degradation in performance needs to be predicted by the semiconductor manufacturers to understand the reliability of their devices. JEDEC defined several standard reliability tests for the semiconductor industry to aid manufacturers in determining the reliability of their semiconductor devices. During these reliability tests, only a certain number of devices (also called *sample size*) across different lots are stressed under elevated environmental conditions of temperature and voltage. These conditions aid to accelerate certain aging mechanisms, so that the minimum reliability of that device, under nominal operating conditions, can be predicted. In general, such tests are termed *reliability tests.*

All the reliability experiments performed during this research work were conducted at high temperatures relative to ambient, while the voltages remain at their nominal levels. The experiment setup was aimed to emulate the HTOL test conditions, wherein the duration of the experiments was targeted for 1,000 hrs with a temperature of 125 ^{o}C. However, to measure the consistency of this approach across diverse temperatures, as discussed in Chapter 1, the experiments were carried out at temperatures ranging from 33 ^{o}C to 145 ^{o}C.

There are numerous challenges to achieve the task of conducting a reliability experiment that remain uninterrupted for 1,000 hrs. A few examples are:

1. An interruption in power supply, or stress signal, even for a couple of

seconds, can let the test chip go under unanticipated recovery which can affect our measurements and subsequent analysis. To overcome this a UPS battery backup that also provides power surge protection was installed.

2. An inconsistent distribution of heat on the test chip can change the value of the measurement parameter. This remains a major limitation of our TYNQ platform for temperatures during which most of the SHEs remain OFF.
3. The 1,000-hour capture and storage of degradation data can lead to memory overflow which can potentially freeze the experiment. This was overcome by setting up an appropriate number of hourly measurements and storing them into a concise file format that requires only a fraction of MBs of space.

The next section discusses a few crucial components of the reliability experimental setup, and how they come together to provide a structure for the experiment. The subsequent section will describe the final execution flow of the experiment.

3.2 Crucial Components for a Reliability Experimental Setup

All the reliability experiments conducted during this research were automated. Once an experiment was launched, there was practically no control over it. This was purposefully done because the test needs to run uninterrupted for at least 1,000 hrs to emulate HTOL test conditions. For that matter, human intervention during the experiment needed to be minimized, so that all our experiments could be a quasi-exact replicas of one another. The automation of experiments requires the combined support of software [SW] and hardware [HW].

3.2.1 Power Supply with Battery Back-up [HW]

Since the reliability experimental setup cannot tolerate any kind of power supply interruption(s), it is necessary for it to possess a supply containing a battery back-up, preferably with surge protection. 1,000 hrs is equivalent to 42 days, which means users must be aware of any planned power outage(s) in the one-and-one-half month window, beginning from the time when the experiment is launched.

UPS batteries are a good fit for short-term (several hours of) supply. Thus, the "CyberPower 1500VA 10-Outlet UPS Battery Back-Up (LX1500GU-FC)" was adopted for the research work. It was estimated that it could support two of the reliability experiments simultaneously for $\sim$5 hrs. In the event of a power interruption, it would have been able to switch over to its battery supply within 4ms, which we felt to be sufficiently rapid for the test platform to be completely unaware of any variety of power interruption. This was verified by manually unplugging the UPS from the power-line during a trial experiment. The UPS device also supported power surge protection, which was crucial since an unprotected power surge has the potential to damage a test chip. In fact, one of the initial experiments was not only interrupted but the test chip likewise damaged due to a power surge event in the facilities. That experiment along with the rest, and their results, are discussed in the subsequent chapters.

3.2.2 Self-heating Module [HW]

To raise the die temperature to conduct thermal stress experiments, the *Self-heating Module* of the TYNQ platform was employed. The module's SHEs generate the required amount of heat to raise the temperature to the desired level (with a maximum achievable level of 145 ^{o}C with 2,304 SHEs), and the on-chip XADC (comprising a temperature sensor) serves as a feedback parameter to regulate the temperature throughout the experiment. A total of 2,304 SHEs was implemented, as displayed in Figure 4.4, (36 *SHE_blocks* and 64 SHEs per block), along with their respective control circuitry that serves to turn the individual blocks on or off. The combination of [*36 * 64*] was estimated by following a trial-and-error method. Although there are multiple possible combinations to achieve the same number of SHEs, the [*36 * 64*] combination was found to provide a stable temperature for the given temperature regulation function: the *fix_temperature* of the **TestChip Driver** module. The *fix_temperature* function enables the user to automate the process of temperature regulation during an experiment.

3.2.3 Circuit Under Test (CUT or BTI Sensor) Module [HW]

To quantify the impact of thermal stress, a ring oscillator (RO)-based architecture was used, as illustrated in Figure 3.1. This is a modified version of the TYNQ's hardware **RO Module**. This architecture is similar to M. Naouss *et al.*'s [30], with the addition of an on-chip digital *Frequency Counter* that monitors the frequency of the **CUT** throughout the experiment, thus aiding in the experiment's automation. The free-running frequency of this RO is a reference parameter against which the degradation (aging) degree is measured.

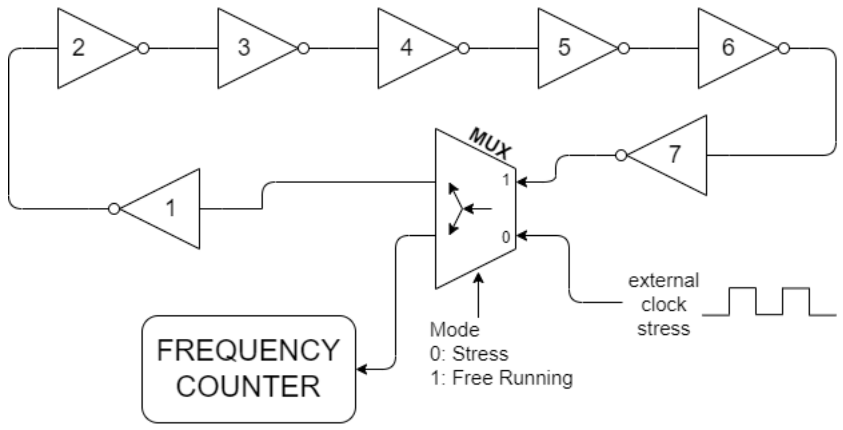


Figure 3.1: Circuit Under Test (CUT) Architecture.

The number of inverter stages in the **CUT** was chosen to be seven since the size of the configurable logic blocks (CLB) is limited to eight LUTs per CLB. Longer **CUTs** require multiple CLBs which increases the routing length for inter-CLB connections, thus adding more unwanted elements to this BTI sensor circuit. Figure 3.2 shows the implementation of a single **CUT** in a CLB.

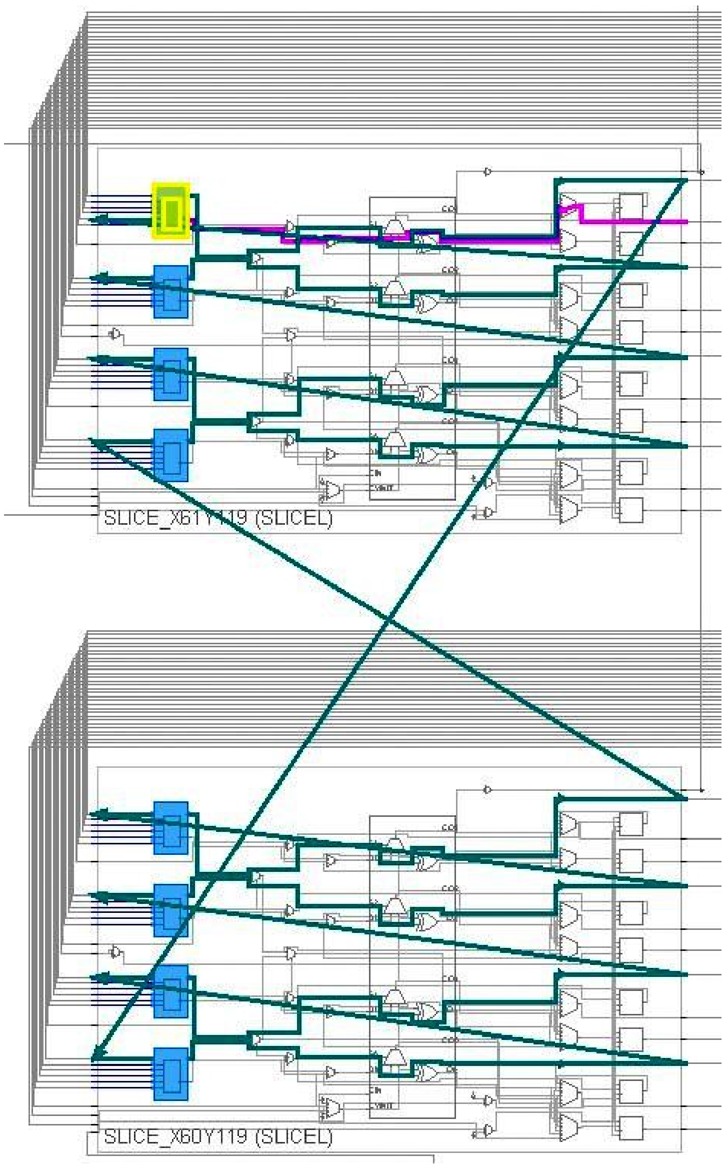


Figure 3.2: The implementation of the **CUT** stages on a CLB. The BLUE blocks are the inverter stages. The YELLOW block is the MUX stage. The GREEN connections are the routing nets. The PINK net connects the MUX stage to the *Frequency Counter*.

Additional functional and implementation details of the **CUT** are as follows:

- **Inverter #**: Each inverter is implemented on the LUT6 (6-input 1-output) described earlier in the TYNQ's **RO Module**.
- **MUX**: The MUX stage is implemented on LUT6_2. It controls the two modes of operation of **CUT**: STRESS and MEASUREMENT. In the STRESS mode, the entire chain of seven inverters oscillates at the external clock's frequency whereas in the MEASUREMENT mode CUT is left to run at its free-running frequency. MUX also duplicates the selected input signal to two nets: one returns to Inverter #1 and the other to the *Frequency Counter*. Appendix E contains the Verilog instantiation of this MUX.
- **"external clock"**: This is a clock signal generated from on-chip the FPGA using *ZYNQ7 Processing System -> PL Fabric Clocks -> IO PLL*. The clock signal performs as an external clock/AC stress signal to the **CUT**. The purpose of this clock signal is to allow for a controllable AC stress signal.

The *Mode* signal to the **CUT** was controlled from the modified TYNQ's **TestChip Driver** module, discussed in the latter sections, which further aided in automating the experiment. The IP of the **CUT Module** can be found in the following GitHub repository: `https://github.com/quanta7/CUT-Circuit-Under-Test/tree/main/ip_repo`.

3.2.4 Python-Based Script *experiment_flow.py* [SW]

To automate the tests, the **TestChip Driver** module (one of the TYNQ's software modules), was extended to include some important features pertaining to the requirements of the reliability experiments, and was re-named the *experiment_flow.py* because it controlled the entire flow of the experiment. Some important features defined in the script that are related to the experiment are as follows:

Temperature Regulation

The *fix_Temperature* function defines the temperature regulation feature by controlling each of the *SHE blocks* independently. This feature takes input (*XADC_temp()*) from the built-in temperature sensor and enables/disables each of these SHE blocks to control the total heat generation so as to regulate the temperature during the experiment. Appendix F contains the *fix_temperature* function script that executes this task. Figure 3.3 illustrates the temperature profile achieved during the experiments that used this feature to regulate die temperature.

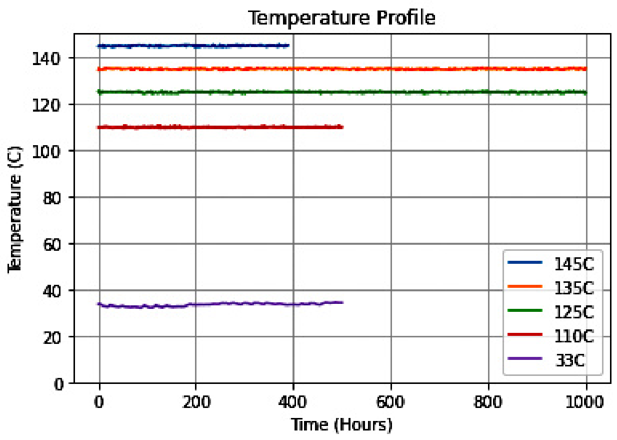


Figure 3.3: Graph illustrating the measured temperature profile from the results obtained during our experiments that used the *fix_temperature* function to regulate die temperatures to the desired temperature level. During these experiments, the temperature level was found to be within $\pm 1^{o}$C of the desired level.

CUT Control

Recall that the **CUT** shown in Figure 3.1 has two modes of operation controlled via the *Mode* signal. The python script (*experiment_flow.py*) has a control over this particular signal for each of the **CUTs** implemented on the test chip. This handle is provided by the *en_feedback* function (added to the **TestChip Driver**), which allows the user to switch between both the modes of operation. The function definition below shows, how the *Mode* signal for all the 20 **CUTs** inside a **CUT Module** can simultaneously be either set to '1' or '0'.

```
def en_feedback(self, mode):
        if mode:
            self.RO2_0.write(0x00000000, 1048575)
            #self.block_name.write(address,value [2^20 - 1])
        else:
            self.RO2_0.write(0x00000000, 0)
            #self.block_name.write(address,value)
```

Read Frequency

There are more than 100 **CUTs** implemented in every experiment on each FPGA, that simultaneously undergo degradation, and the digital frequency counters continuously monitor their frequency. This feature reads the frequency value stored simultaneously in all the counters and stores it for further formatting and transmission. Appendix G contains the *read_multi_RO* function script that reads the oscillation frequency of the input **CUT**s.

Data Format and Storage

Apart from the frequency of **CUTs**, we also measured various internal voltage levels (those that are accessible to the user), die temperatures and the time instants of measurement. At each hour of the experiment, this information is captured, organized and stored in a file in a pre-defined format. The files generated were stored locally on the microSD card of the PYNQ-Z1 board. The *record* function in Appendix H performed this job by taking some basic inputs: the FPGA bitstream (an implementation image of all the circuits), durations of measurement, sampling step sizes, numbers of **CUT**s, and the desired temperature levels, respectively. The sampling step size duration was set according to the time consumed by the ZYNQ processing system to measure, hold and transfer the *frequency counter* data. This prevented the system from entering into standby during every call of data measurement and transfer.

Dispatch Email

This feature sends an email containing an attachment of the new files generated every hour. This is primarily done to prevent data loss during unforeseen scenarios. The function *sendemail* in Appendix I performs this task. Potential errors, including discontinuity in internet connections, could freeze this function (and the experiment) if they coincided with the time instant when this function was called. For its seamless operation, errors rising due to such scenarios should be handled in the function definition. This function needs a sender's Gmail address and password prior to the launch of every experiment. However, with the upcoming changes from May 30, 2022 onward in Google's Security and Privacy Settings, a third-party application such as this PYNQ framework on which the test platform was built, would no longer be able to sign into a Google account using only the email address and password. Thus, a user should modify the function in accordance to support their respective organization's webmail services.

3.3 Structure of the Experimental Setup

The entire physical structure of the experiment is shown in Figure 3.4. The PYNQ board always received its power supply from the UPS to remain immune from any power interruption(s) and power surge(s). All bitstream files from the Vivado software were stored on the microSD card of the PYNQ board prior to the launch of the experiment. The *experiment_flow.py* controlled the stress experiment on FPGA while simultaneously communicating with a microSD card (embedded on the PYNQ-Z1 board) to store the data, and the PYNQ board's web connectivity to transmit the data over the web.

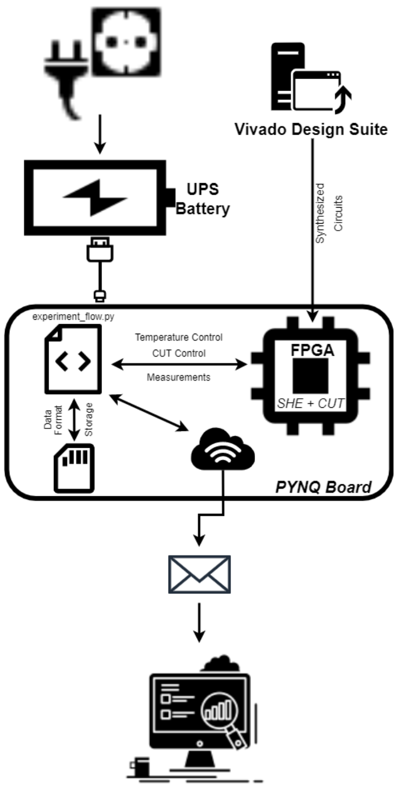


Figure 3.4: Complete Structure of the Experimental Setup.

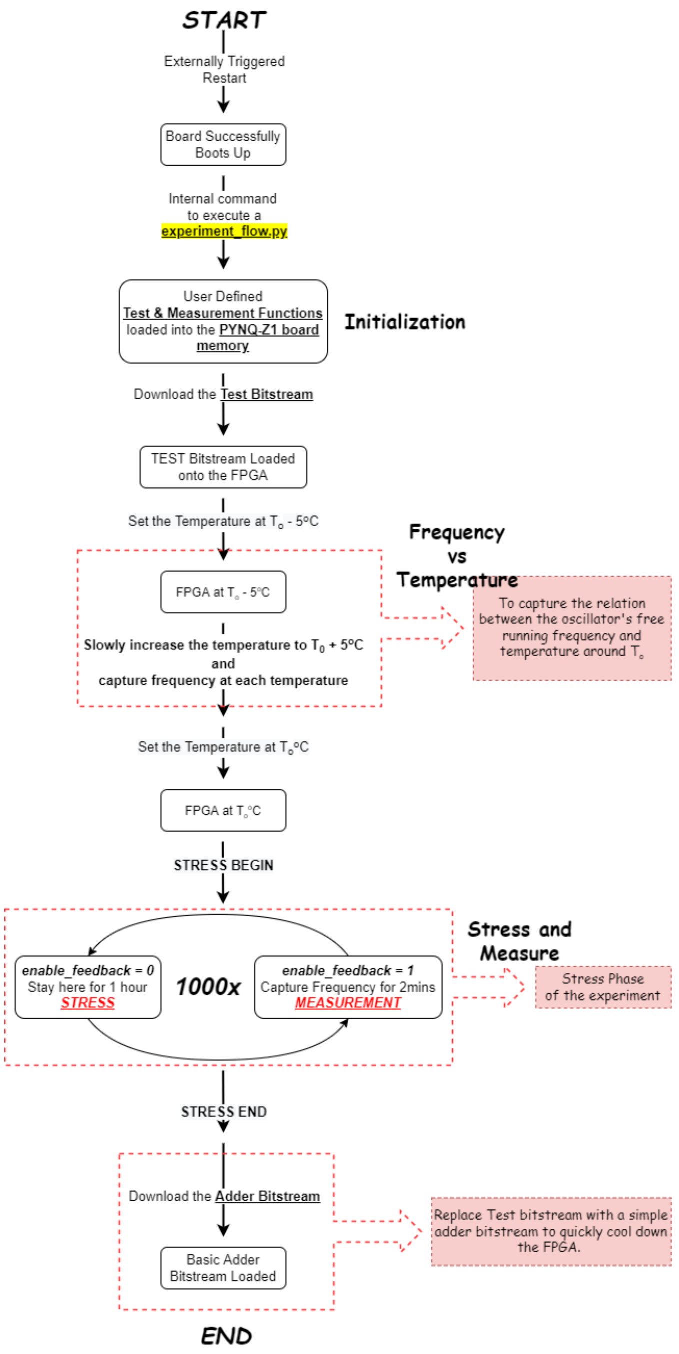

START
Externally Triggered Restart
Board Successfully Boots Up
Internal command to execute a experiment_flow.py
User Defined Test & Measurement Functions loaded into the PYNQ-Z1 board memory
Initialization
Download the Test Bitstream
TEST Bitstream Loaded onto the FPGA
Set the Temperature at T_o - 5°C
Frequency vs Temperature
FPGA at T_o - 5°C
Slowly increase the temperature to T_0 + 5°C and capture frequency at each temperature
To capture the relation between the oscillator's free running frequency and temperature around T_o
Set the Temperature at T_o°C
FPGA at T_o°C
STRESS BEGIN
Stress and Measure
enable_feedback = 0 Stay here for 1 hour STRESS
1000x
enable_feedback = 1 Capture Frequency for 2mins MEASUREMENT
Stress Phase of the experiment
STRESS END
Download the Adder Bitstream
Basic Adder Bitstream Loaded
Replace Test bitstream with a simple adder bitstream to quickly cool down the FPGA.
END

Figure 3.5: Thermal Stress Test Execution Flow Chart.

3.4 Flow of the Reliability Experiments

The earlier sections briefly discussed various software and hardware components which together made the experimental automation feasible. This section describes the automated executional flow of the experiments as represented in Figure 3.5. The entire flow of the experiment was divided into three phases: *Initialization*, *Frequency vs Temperature* and *Stress and Measure*. The intermediate phase of *Frequency vs Temperature* was not useful if the temperature regulation feature performed to its expectations. Note that the entire flow of the experiment was open-loop, i.e. at each step in the flow, the system assumed that all the previous step(s) had been successfully executed and never attempted to verify the success/failure of the previous step.

1. **Initialization:** In this phase, the *experiment_flow.py* script was loaded into the PYNQ-Z1 memory. Since this script controlled the entire flow, from this point forward, this python script became the sole driver of the experiment. Subsequently, the test bitstream (FPGA circuits implementation image) was downloaded onto the FPGA. A successful download meant all the SHEs and **CUTs** were in place. Following this, a command from the script began to increase the temperature of the chip to T_0-5 °C (T_0 being the stress temperature) by using its **Temperature Regulation** feature.

2. **Frequency vs Temperature:** After the temperature reached [T_0-5 °C] (T_0 being the stress temperature), it was raised in steps of 2°C (up to T_0+5 °C) to measure the frequency of **CUTs** at every step. The measured data was stored, and its copy also transmitted over the web using the **Dispatch Email** feature. This measurement was performed to capture the relationship between frequency and temperature, which could be used, if necessary, during the data analysis. Later, the chip temperature was achieved $\sim T_0$°C. The portion of the *experiment_flow.py* that executed Phases 1 and 2 is shown in the Appendix J.

3. **Stress and Measure:** In this phase, the **CUT** was stressed for an hour at the *external clock* frequency, and then its free-running frequency was measured over a window of 2 minutes (measuring multiple times to increase the precision) using the **Read Frequency** feature. Subsequently, the measured data was stored locally using the **Data Format and Storage** feature, and also transmitted over the web.

This was repeated 1,000 times. This switching between different modes was possible with the help of the **CUT Control** feature. The python script that performed this repetitive operation and regularly dispatch the data as email attachments is shown in Appendix K.

3.5 Conclusion

In this chapter we examined various crucial HW and SW components of the experimental setup, and how they combine to build the complete structure of the reliability experiment. We also looked at various features of these components that served to make the automation of the 1,000-hour-long experiment feasible. With the reliability test platform in place and by creating a structure based on it, we conducted several thermal stress experiments whose results and analyses are discussed in the next chapters.

Chapter 4

Thermal Stress Experiments

4.1 Aim

Previous chapters looked at the reliability test platform, its various hardware and software components that provided the necessary features to run the experiment over a long duration, and the structures of the experiments that enabled automation and uninterrupted flow. In this chapter, all the reliability experiments conducted and their measurement results are reported.

4.2 General Test Setup

Figure 4.1 portray the physical setup of the reliability experiment. This setup was placed in one of UBC's ECE laboratories. The PYNQ-Z1 board was connected to the internet via a LAN (Local Area Network) connection. The board was kept within a glass chamber filled with fibreglass. Fibreglass has a very high melting temperature (≈ 1500 °C). This property makes it a viable option for our setup for use in insulation, causing the heat to remain trapped and allowing the test chip to easily increase to the desired temperature. In the case of any electric failure on the board, the fibreglass would not have been melted by the resulting transient excessive heat.

During each experiment, a total of six batches of the **CUT Module** were implemented on the FPGA, with each batch consisting of 20 **CUTs**. Each batch received a unique *external clock* signal that remained unchanged throughout the experiment. For the six batches, there were six different clock signals covering a wide range of frequencies: 225 MHz, 100 MHz, 6.25 MHz, 390 KHz, 24 KHz and 12 KHz. This was done primarily to capture the contribution of another aging mechanism HCI, since HCI occurs during transistor switching, implying that batch stressed at a higher clock frequency would be expected to show significantly higher HCI-induced degradation. To render all these frequencies a basic clock divider circuit was implemented us-

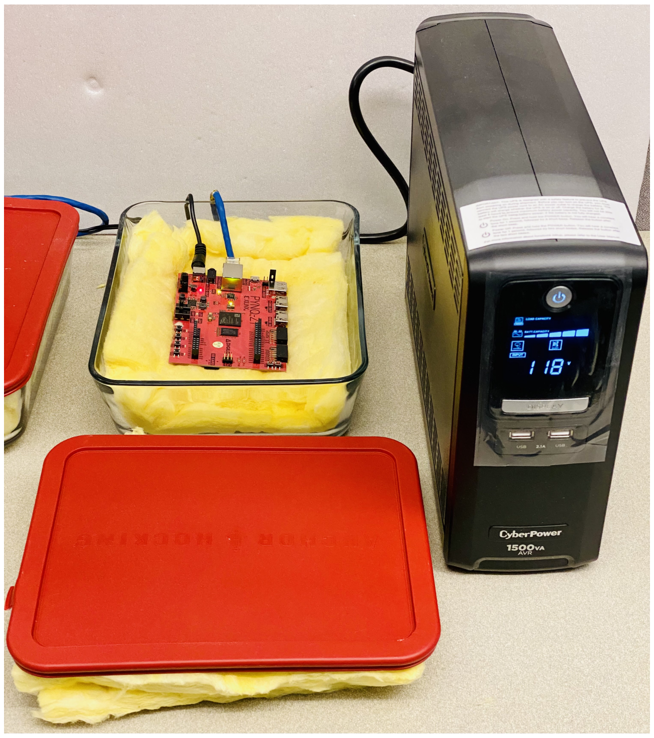
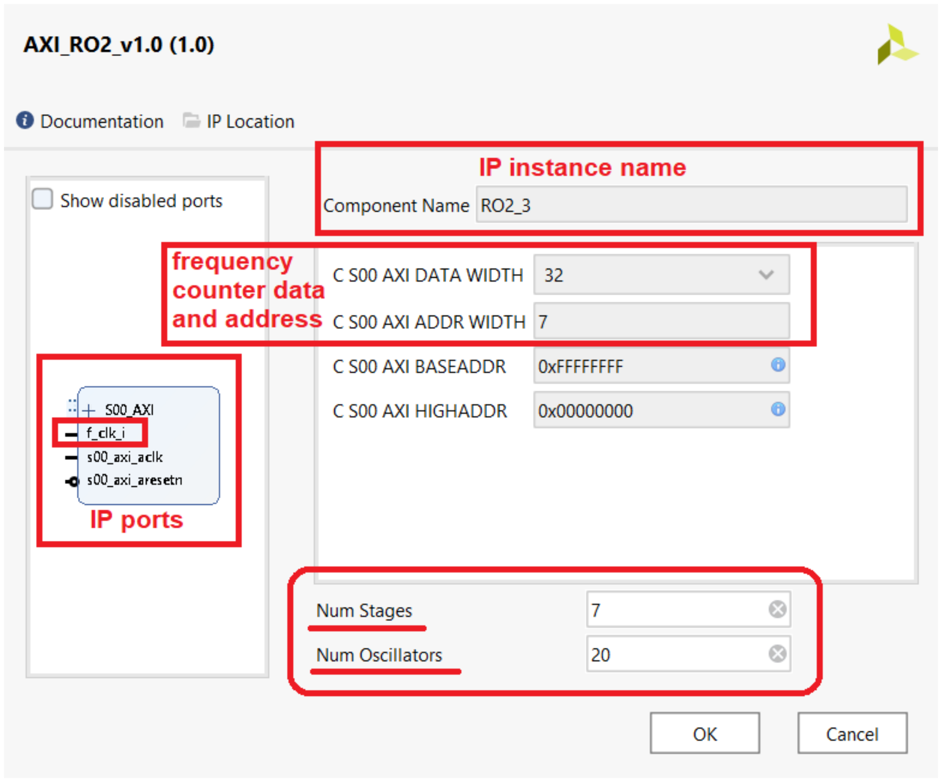


Figure 4.1: An image showcasing our experiment setup in the laboratory. The PYNQ-Z1 board was placed within a glass container filled with fibreglass (yellow). *CyberPower 1500VA* is the UPS providing power-backup and surge protection. Prior to the launch of an experiment, some fibreglass was placed on the board, along with the container's lid (shown in red color).

ing D Flip-Flops (D-FFs). Its Verilog instantiation source code is provided in Appendix L.

Figure 4.2 displays the Vivado IP customization window of an instantiation of the **CUT Module** with the name "RO2_3". The IP ports in the customization window has an additional port of *f_clk_i* which does not exist in the IP ports of the **RO Module** (Figure 2.5). The *f_clk_i* port was introduced to feed the **external clock** signal to all the **CUTs** inside the **CUT Module**. The block design of the entire system on the FPGA is illustrated in Figure 4.3 and its placement on the chip is highlighted in Figure 4.4.



AXI_RO2_v1.0 (1.0)

Show disabled ports

Component Name RO2_3

C S00 AXI DATA WIDTH 32

C S00 AXI ADDR WIDTH 7

C S00 AXI BASEADDR 0xFFFFFFFF

C S00 AXI HIGHADDR 0x00000000

Num Stages 7

Num Oscillators 20

Figure 4.2: The Vivado IP Customization window: **CUT Module** customization of its number of stages (*Num Stages*) and the total number of **CUTs** (*Num Oscillators*).

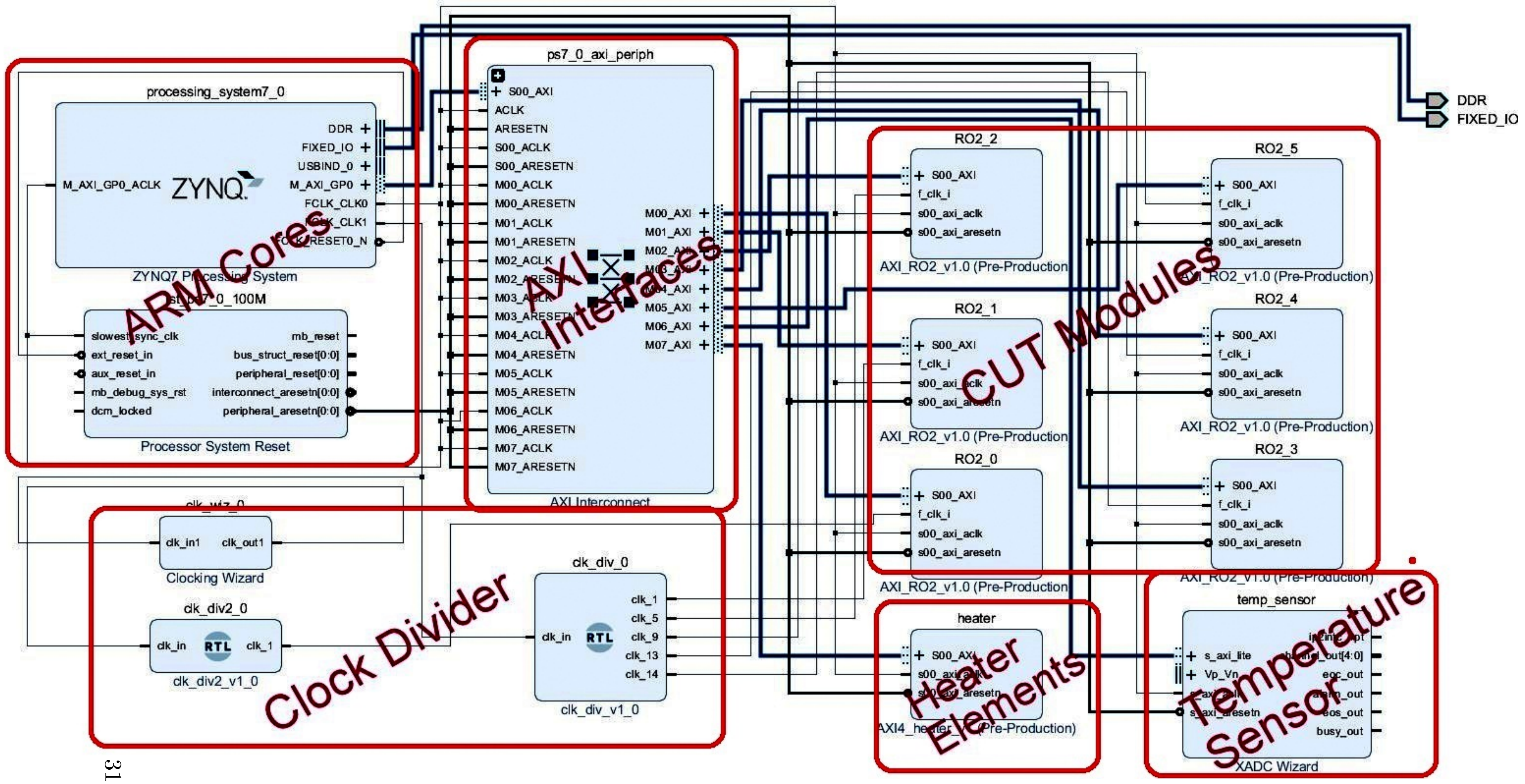


Figure 4.3: The Vivado Block Design of the system implemented on the Zynq XC7Z020-1CLG400C. A highlighted instantiation of six **CUT Modules**, the **Self-heating Module** (HEATER ELEMENTS) and the Temperature Sensor (XADC).

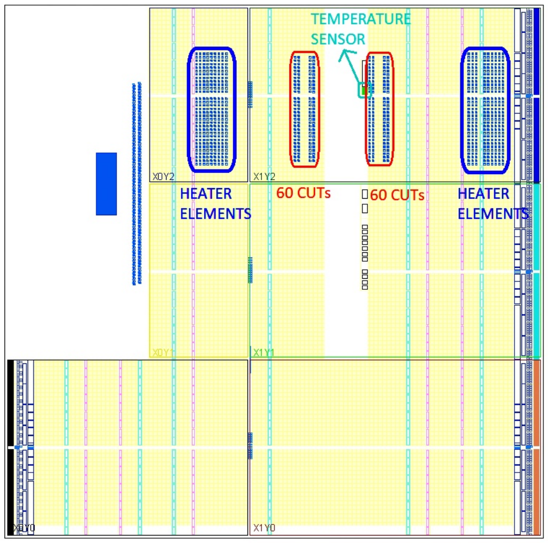


Figure 4.4: The Zynq XC7Z020-1CLG400C layout. A highlighted placement of the 120 **CUTs**, the heater elements and the XADC (Temperature Sensor).

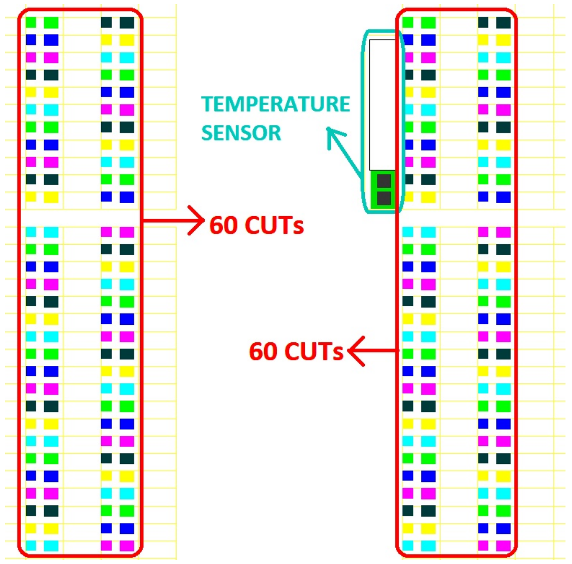


Figure 4.5: The Zynq XC7Z020-1CLG400C layout. A zoomed image of the 120 **CUTs** placement, showcasing the uniform distribution of 20 **CUTs** of each of the six batches.

4.3 Experiments

This section describes the measurements obtained in the thermal stress experiments, each conducted on a different chip. The fixed test execution flow of Figure 3.5 was maintained for all the experiments. The duration of each of those experiments is shown in Table 4.1. The stress temperature of 145 °C was observed to be the maximum achievable temperature on the chip. The minimum ambient temperature that we could achieve by employing a mini-refrigerator was 0 °C, but due to the unavoidable heating of the chip, the minimum attainable die temperature was measured at ∼33 °C. The temperature level of 125 °C was chosen in accordance with the HTOL test conditions. The 110 °C and 135 °C stress temperature experiments were added to expand the scope of the experiments to study the activation energy (E_a) of the **BTI** aging mechanism. Although it was proposed that all the experiments be conducted for 1,000 hrs, some needed to be curtailed to shorter durations because the test setup was concurrently being modified. Experiments #1 and #2 were the longest experiments, and were conducted when the reliability test platform was in a more stable state than it had been during the Experiments #3, #4 and #5. The measurements from each experiment are described in this section. A detailed analysis of those results will be covered in the subsequent chapter.

Table 4.1: The Duration of the Thermal Stress Experiments

EXP.	**Experiment**	
No.	***Stress Temperature***	***Stress Duration***
#1	135 °C	1,000 hrs
#2	125 °C	1,000 hrs
#3	110 °C	500 hrs
#4	33 °C	500 hrs
#5	145 °C	390 hrs

4.3.1 135 °C and 125 °C Thermal Stress Experiments

During the experiment, the free-running frequency (F) of all the 120 **CUTs** (six batches each consisting of 20 **CUTs**) were measured along with various internal voltages and temperature of the chip. Table 4.2 illustrates the *[*$\frac{\Delta F}{F}$ *%]* calculations of 20 **CUTs** of a batch labelled "**RO2_3**" from $T = 0hr$ to

$T = 500hr$.

$$\frac{\Delta F}{F}\ \% = 100 * \frac{F_T - F_{0hr}}{F_{0hr}}\ \% \tag{4.1}$$

Table 4.2: Frequency degradation of 20 **CUTs** of **RO2_3** batch at 135 ^oC

Time	**#1**	**#2**	**...**	**#20**
T = 0 hr	190.68MHz	191.28MHz	...	190.64MHz
T = 500 hr	188.28MHz	188.80MHz	...	188.07MHz
$\frac{\Delta F}{F}$ %	-1.258%	-1.296%	...	-1.348%
Average	-1.202%*			

*Plotted in Figure 4.6

The *Average* value represents the degradation in frequency in the **RO2_3** batch when the experiment has ran for 500 hrs. Similarly measuring the degradation from time zero to every hour of the experiment results in a frequency degradation curve, as shown in Figure 4.6.

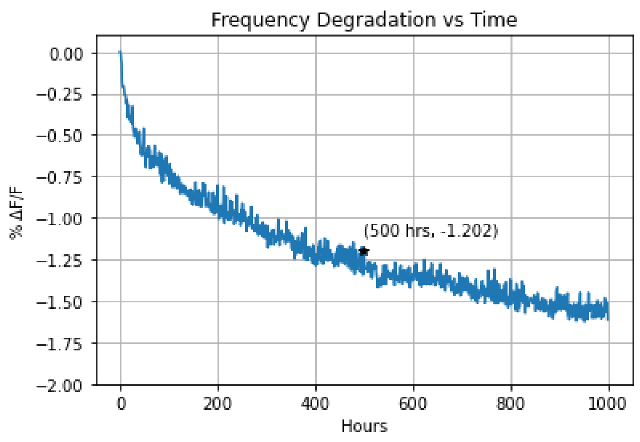


Figure 4.6: Frequency degradation of "RO2_3" (390KHz) batch at 135 ^oC.

Since there were six batches, there were six frequency degradation curves. The frequency degradation curves of all the six batches are shown in the Figure 4.7. To perform any analysis on the frequency degradation data, it was essential to reduce the noise in the data. The noise in the curves were filtered out by applying the *Butterworth* filter available in the *Signal* function of *SciPy* (an open-source library in Python). After filtering out the noise, the resultant final frequency degradation curves are shown in the Figures 4.8 and 4.9.

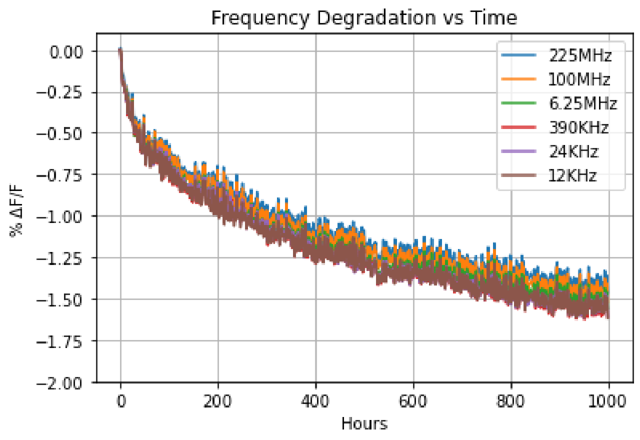


Figure 4.7: Frequency degradation of the six batches at 135 °C (unfiltered).

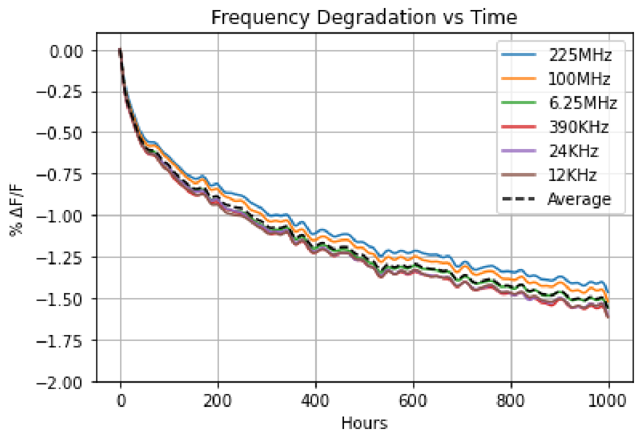


Figure 4.8: Frequency degradation of the six batches at 135 °C.

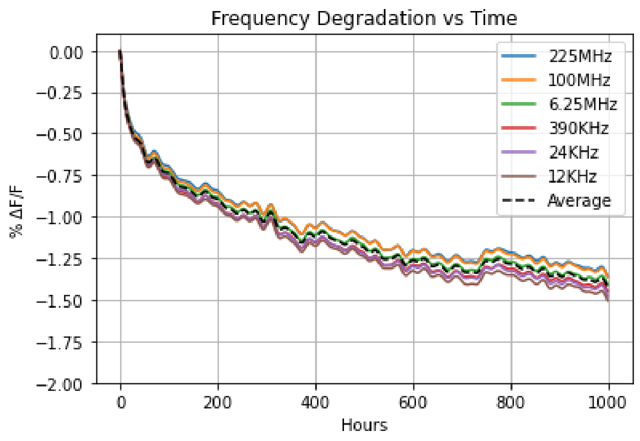


Figure 4.9: Frequency degradation of the six batches at 125 °C.

4.3.2 110 °C Thermal Stress Experiment

The duration of the experiment was limited to 500 hrs as the test setup was concurrently being developed. The frequency degradation of the six batches during the experiment are shown in Figure 4.10. The measured frequency degradation profile in the experiment has a couple of large crests and troughs. This is because of inconsistent heat distribution across the chip during the experiment. The **Temperature Regulation** feature only targets the temperature sensor point on the chip (Figure 4.4) to regulate the temperature but does not take into consideration the distribution of heat across the chip. Since the 110 °C temperature level requires very few SHEs ($\ll$ 2,304 SHEs) to remain on and these SHEs were randomly selected by the **Temperature Regulation** feature, the temperature did not remain consistent in all the areas of the chip. As a consequence the distribution of heat throughout the test chip changed, giving rise to temperature-induced drift in the frequency of the **CUTs.** This observation marked a limitation of the *Self-heating Module*.

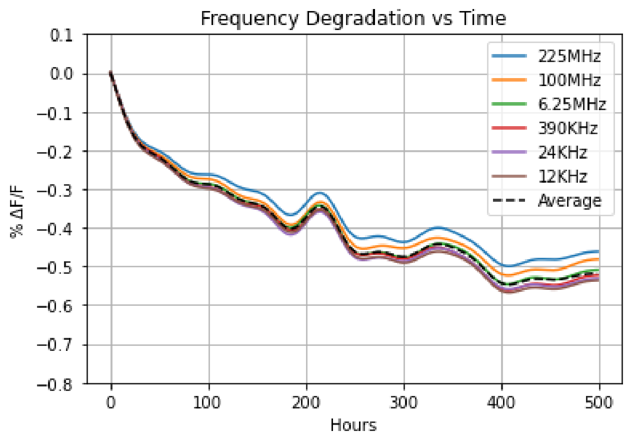


Figure 4.10: Frequency degradation of the six batches at 110 °C.

4.3.3 33 °C Thermal Stress Experiment

The duration of the experiment was also limited to 500 hrs. It was aimed at enhancing the contribution of the HCI aging mechanism to the circuit degradation, which could be achieved by lowering the stress temperature. However, the reliability test platform was designed to elevate, not reduce the chip temperature. Thus, it was required to force the desired temperature using external equipment. Considering the duration of the experiment (>20 days), a mini-refrigerator was employed that could cool down the ambient air to 0 °C at the most. Therefore, with the ambience verified at 0 °C and the minimum unavoidable heating of the test chip, the minimum achievable chip temperature was found to be ∼32-34 °C. Note that the expectation regarding the refrigerator was that the ambient temperature would be maintained exactly at 0 °C, but the fridge failed to maintain that level accurately, explaining why the temperature during the experiment fluctuated, leading to more crests and troughs on the frequency degradation profile. The frequency degradation results are shown in Figure 4.11.

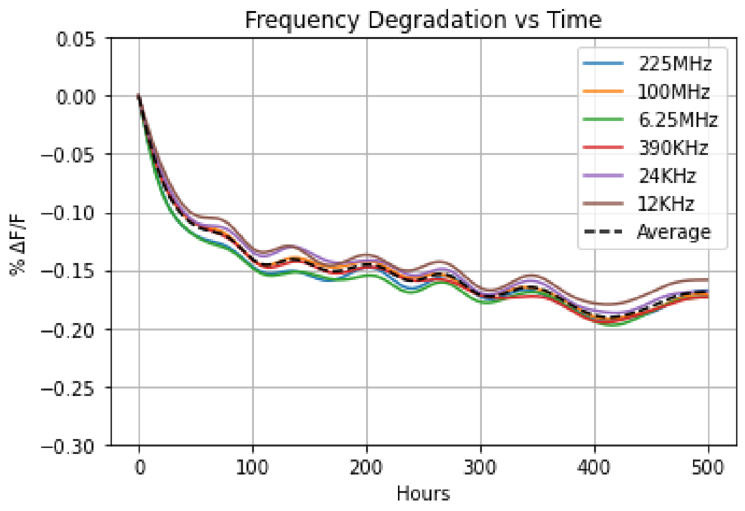


Figure 4.11: Frequency degradation of the six batches at 33 °C.

4.3.4 145 °C Thermal Stress Experiment

The experiment was initially to be conducted with the test chip subjected to the highest possible temperature. However, a high-temperature requirement necessitates the activation of a high number of SHEs to produce the required amount of heat, which further leads to a high degree of power consumption. The PYNQ-Z1 board on which the test chip (FPGA) was mounted was equipped with a power regulator that disabled the power supply to the chip were the power consumption to exceed a certain limit. An optimal position retaining the test chip within its power limits while allowing the subject to function at a desirable stress temperature was revealed to be ∼146 °C. The experiment was designed to last for 1,000 hrs but a power surge in the facilities caused the experiment to shut down at the 391st hour. Consequently, the results collection for this experiment was limited to 390 hrs.

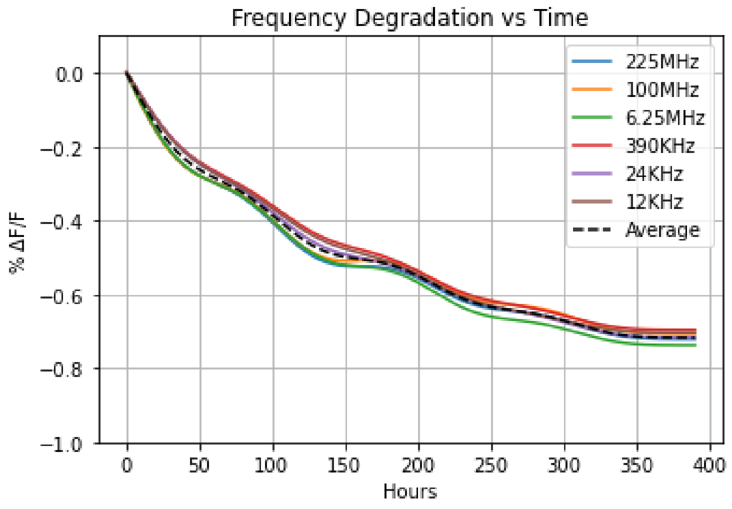


Figure 4.12: Frequency degradation of the six batches at 145 °C.

4.4 Conclusion

This chapter covered in detail the setup and measurement results of each of the thermal stress experiments. The *Self-heating Module* of the TYNQ platform, was found to maintain the device's temperature within ±1 °C. However, it did not perform very well in maintaining heat distribution across the chip at temperatures where most of the SHEs remained turned off. This marked a limitation for this module and scope for improvement. The next chapter describes various analyses performed on the measurement results and subsequent conclusions.

Chapter 5

Results Analyses

5.1 Introduction

Previous chapter highlighted the reliability experiment setup and frequency degradation results obtained from each of the thermal stress experiments. This chapter describes the analysis performed on those degradation results. Recall that the primary aim of these experiments is to determine if the degradation data collected during the initial duration of the experiment can enable us to predict the degradation occurring during the remainder of the experiment. To perform a prediction, the first step is to develop a circuit-level degradation model that mathematically describes the **CUT's** frequency (the reference parameter to measure degradation) degradation as a function of time. This model will be termed the **CUT** *Frequency Degradation Model.* The second step is to fit that model with frequency degradation data collected over a relatively shorter time frame to compute the model's parametric constants. Henceforth, the term *prediction data* will refer to a set of frequency degradation data from the initial duration of an experiment which will be used to perform prediction.

The analysis will begin with the results obtained from Experiments #1 and #2, since they are the longest among the five experiments being discussed and their frequency degradation curves have strict monotonic exponential decay profiles. Later, the same analysis will be performed on the rest of the shorter duration experiments.

5.1.1 CUT Frequency Degradation Model

The **CUT** is made up of digital gates whose individual propagation delays (τ_d), following [18], can be approximated as:

$$\tau_d \sim \frac{C_L V_{dd}}{I_d} \propto \frac{C_L V_{dd}}{(V_{dd} - V_T)} \tag{5.1}$$

where C$_L$ is the gate's output capacitance, V$_{dd}$ is its supply voltage, and V$_T$ is the threshold voltage of the transistor. If V$_T$ changes by ΔV_T then the change in the propagation delay of the gate will be:

$$\Delta\tau_d \propto \frac{\Delta V_T}{(V_{dd} - V_T)}\tau_d \tag{5.2}$$

Since the free-running frequency of the CUT, F, is inversely proportional to the propagation delay of its digital logic gates, i.e.

$$F \propto \frac{1}{\tau_d} \tag{5.3}$$

A $\Delta\tau_d$ change in the propagation delay will result in:

$$\Delta F \propto -\frac{\Delta\tau_d}{\tau_d^2} \propto -\Delta V_T \tag{5.4}$$

i.e. for the CUT, the degradation in F is directly proportional to the degradation in V_T of the transistor. For the degradation in V$_T$, X. Li *et al.* in their article [34] model the V$_T$ shift as:

$$\Delta V_T = \Delta V_{T_{max}}[1 - e^{-(t/\tau)^{\beta}}] \tag{5.5}$$

Combining equations (5.4) and (5.5), the **CUT** Frequency Degradation model can be defined as:

$$\frac{\Delta F}{F} = A[1 - e^{-(Bt)^{C}}] \tag{5.6}$$

where A, B and C are unknown parametric constants.

The parametric constants are determined by fitting the model with the *prediction data* (experimental data collected over a relatively shorter time frame), as described in the following section.

5.2 Analysis I

5.2.1 125 °C and 135 °C Thermal Stress Experiments

The average degradation curves of both these experiments are demonstrated again in Figure 5.1. The figure not only reveal the measured frequency degradation curves; but also *predicted curves* (dashed). The predicted curve describes the size of the *prediction data* which conforms to the **CUT** *Frequency*

Degradation Model to determine its parametric constants and extrapolate them until the end of the experiment. Figure 5.1 illustrates that, on using only the first 100 hrs of frequency degradation data, i.e. 100 points (one set of measurements per hour), to determine the model parametric constants, the resultant model could not follow the *actual* degradation data for the remaining duration of the experiment. The same is observed when 200 hrs of degradation data is used. However, the *200 Hrs* curve is relatively closer to the *actual* curve if compared with the *100 Hrs* curve. However, subsequent to using 400 hrs of data, the resultant curve follows the *actual* curve throughout the experiment, unlike the *100 Hrs* and *200 Hrs* curves, which completely deviate, past their own time marks, from the *actual* curve. This means that some finite period of initial hours of frequency degradation data (400 hrs in this case) possesses sufficient information to accurately forecast the degradation to transpire during the remaining hours of the experiment.

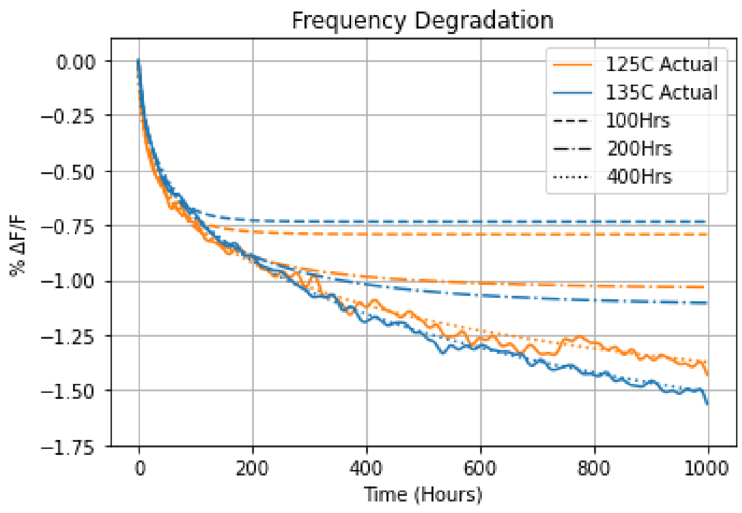


Figure 5.1: The 'Actual' (solid) vs. 3-'Predicted' Degradation curves.

However, this observation does not guarantee the 400-hour time mark, although it indicates the potential information embedded in the initial hours of degradation data that can be exploited using this method to obtain a *predicted curve*, which can give an estimated degradation value during each of the remaining hours of the experiment. To analyze the accuracy of our prediction not at a single time point but for all the time points during our accelerated aging test, a quantitative comparison of the *predicted curve* with the *actual* curve will be described in the subsequent sections. Before that, it is essential to understand the curve-fitting algorithms that were used to generate such *predicted curves.*

5.2.2 Curve Fitting Algorithms

SciPy, an open-source library in Python, provides fundamental algorithms for various optimization problems. An optimization problem of interest here is *Curve Fitting*, because we have a function, given in (5.6), which is expected to fit the frequency degradation curves described in Chapter 4. To perform the curve fitting, the *curve_fit* function under the *scipy.optimize* tool was relevant. The *curve_fit* function facilitated selecting from among the algorithms for curve fitting, namely: *Trust Region Reflective (trf)*, *dogleg (dogbox)* and *Levenberg-Marquardt (lm). Trust Region Reflective* is particularly suitable for large problems, whereas *Levenberg-Marquardt* is efficient for small unconstrained problems.

The results obtained from individually using these algorithms are shown in Figures 5.2, 5.3 and 5.4. By comparing these results, it was found that the *trf* and *dogbox* algorithms rendered almost identical predicted curves (*100 Hrs, 200 Hrs* and *400 Hrs*). This means that the optimization performance of both the *trf* and *dogbox* algorithms is the same for our curve fitting problem. Consequently a user can employ either of these algorithms to perform curve fitting. Moving onto the results obtained from the *lm* algorithm in Figure 5.4, it was observed that this algorithm could only produce the optimal curves for small prediction data sets, such as 100 or 200 hrs, since its predicted curve for large prediction data sets, like 400 hrs, did not fit the *actual* curve. However, the optimal curves rendered by *lm* algorithm for small prediction data sets were also almost identical to that of the *trf* and *dogbox.* Therefore, all the *predicted curves* discussed earlier and those subsequently, were rendered using the *Trust Region Reflective* algorithm, unless otherwise stated.

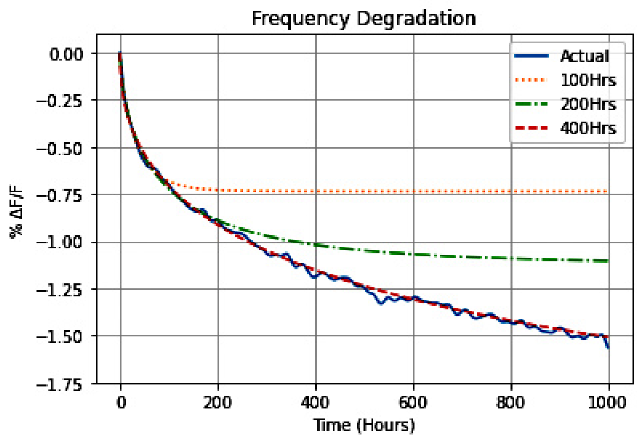


Figure 5.2: *trf* algorithm: 'Actual' vs. 3-'Predicted' Degradation curves.

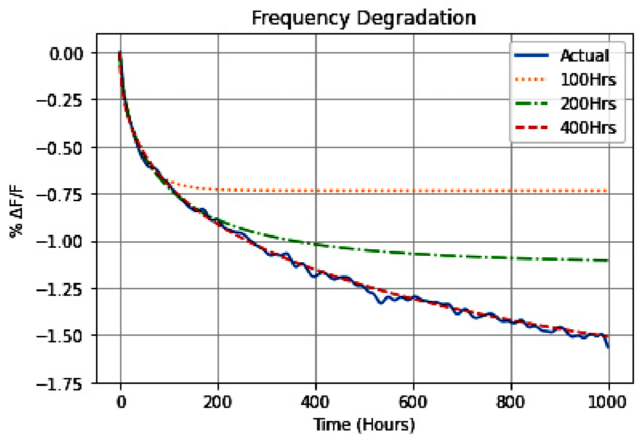


Figure 5.3: *dogbox* algorithm: 'Actual' vs. 3-'Predicted' Degradation curves.

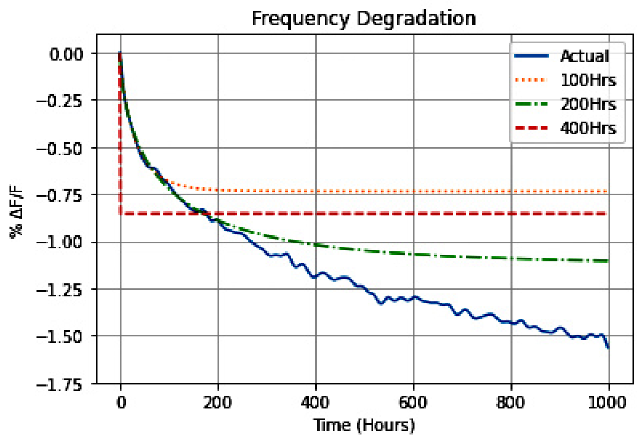


Figure 5.4: *lm* algorithm: 'Actual' vs. 3-'Predicted' Degradation curves.

5.2.3 Discrepancy Between the Predicted and Actual Curve

The term, *prediction discrepancy* will be utilized to quantify the discrepancy between the *predicted* and the *actual curve*. Here, *prediction discrepancy* is defined as the degree of error between the *predicted* and *actual curve* with respect to the *actual curve*. To understand the calculation of *prediction discrepancy*, please refer to Table 5.1, which illustrates the discrepancy between the *100 Hrs* and *actual* curves. Each cell value in the rows "100 Hrs curve" and "Actual curve" represents the respective points on the *100 Hrs* and *Actual* curves, of Figure 5.1 for 135 °C experiment. The "|Error|" represents the relative degree of error at each hour between the values of the two curves. The "Mean[|Error|]" is the mean value of all the errors in the "|Error|" row; i.e., the value of the "Mean[|Error|]" represents the relative degree of mean error between the two curves.

Table 5.1: An Illustration of the Error Calculation Between Two Curves

Curves	**0hr**	**1hr**	**2hr**	**...**	**500hr**	**...**	**999hr**	**1,000hr**
100 Hrs Curve	0	-0.0403	-0.0703	...	-0.7346	...	-0.7346	-0.7346
Actual Curve	0	-0.0286	-0.0569	...	-1.2451	...	-1.5541	-1.5623
|Error|	-	41.01%	23.6%	...	41%	...	52.7%	52.9%
Mean[|Error|]	34.9%*							

*Plotted on the curve in Figure 5.5

Similarly, calculating the *prediction discrepancy* for different *predicted curves* ranging from *1 Hr* to a maximum of *1,000 Hrs* results in a curve revealed in Figure 5.5, where the x-coordinates of a point on the curve denotes the size of the prediction data used to generate the *predicted curve*, and the y-coordinates denotes the value of *prediction discrepancy* between the *predicted* and the *Actual* curve. For example, the value of "Mean[|Error|]" in Table 5.1 would correspond to the point (100 hrs, 34.9%).

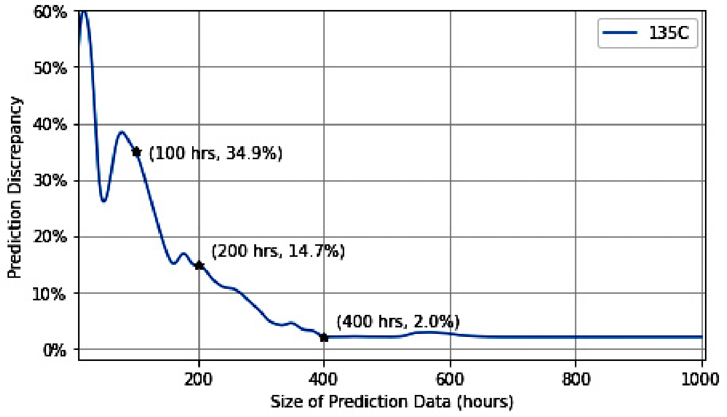


Figure 5.5: Variation in *Prediction Discrepancy* on increasing the size of the *Prediction Data* at 135 ^{o}C. Highlighted points for *prediction data* sets of size 100 hrs, 200 hrs and 400 hrs.

It is evident from Figures 5.5 and 5.6 that, as the *prediction data* size increases, the *prediction discrepancy* decreases, which aligns with our expectations. However, the observation of interest is that the *prediction discrepancy* saturates at close to zero, roughly beyond the 400-hour mark. This means that the necessary information regarding the future degradation of the circuit exists in the degradation data obtained from the first half of the experiment. In other words, had I continued my experiments until the half-time period, I would have accurately estimated the entire degradation until the completion of the experiment. This potentially implies a significant reduction in test time. It is important to highlight here that the *prediction discrepancy* never reaches zero. It saturates to $\sim$2%, even when the entire 1,000 hrs of degradation data is utilized. This is primarily due to the random noise (often called white noise) in the measured data, which arises from various sources. The *predicted curves* that are generated using the *prediction data*, as with the *100 Hrs, 200 Hrs,* and other experiments outlined

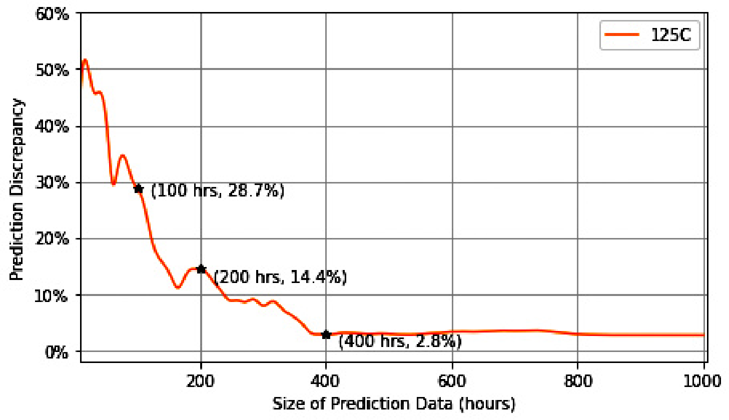


Figure 5.6: Variation in *Prediction Discrepancy* on increasing the size of the *Prediction Data* at 125 °C. Highlighted points for *prediction data* sets of size 100 hrs, 200 hrs and 400 hrs.

in Figure 5.1, follow a noiseless equation. For that reason, we could not achieve zero discrepancy. Thus, for instance, if the saturation level were considered our reference baseline, then the evidential inaccuracy from using the *prediction data* set size of 400 hrs would be nearly zero.

Note that there is nothing of particular relevance in regards to 400 hrs (or the 40% duration of the experiment). But this observation reveals the existence of potential information in the initial hours of the degradation data that can be exploited using this method to predict such accelerated aging test results. As well, during this entire process of prediction, the process corner (strong/typical/weak) of the chip was never taken into consideration, although it was able to accurately predict the amount of degradation. This is explained by the data driven nature of this approach; i.e., the process of aligning the **CUT** *Frequency Degradation Model* with the *prediction data* inherently extracts the information from the chip's process corner and embeds it into the parametric constants, A, B and C. Therefore, regardless of the process corner, this approach would be effective in determining the final degradation of the circuit.

5.2.4 110 °C, 33 °C and 145 °C Thermal Stress Experiments

By performing the same analysis on the frequency degradation data obtained from Experiments #3, #4 and #5, the same trend in *prediction discrepancy* was observed, as shown in Figure 5.8. The saturation level was also found to be in the same range, although consistently somewhat higher than that of Experiments #1 and #2. This is again since the reliability test platform was not as stable during those experiments as it was for the most recent ones, #1 and #2. Table 5.2 highlights the summary of all these thermal stress experiments.

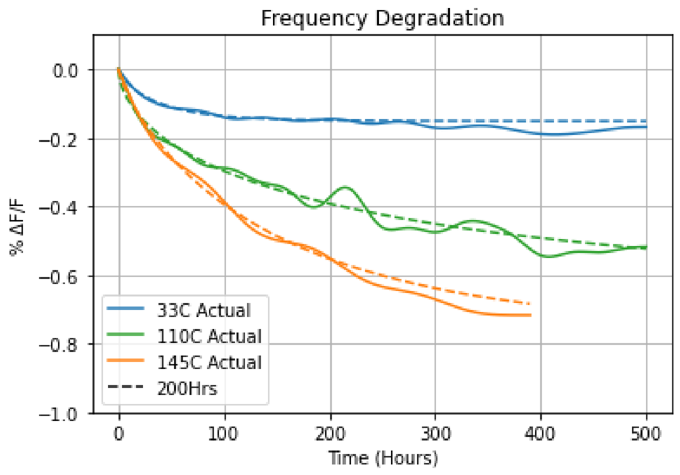


Figure 5.7: 'Actual' (solid) and 'Predicted' ('−−') degradation curves for the 145 °C, 110 °C and 33 °C experiments.

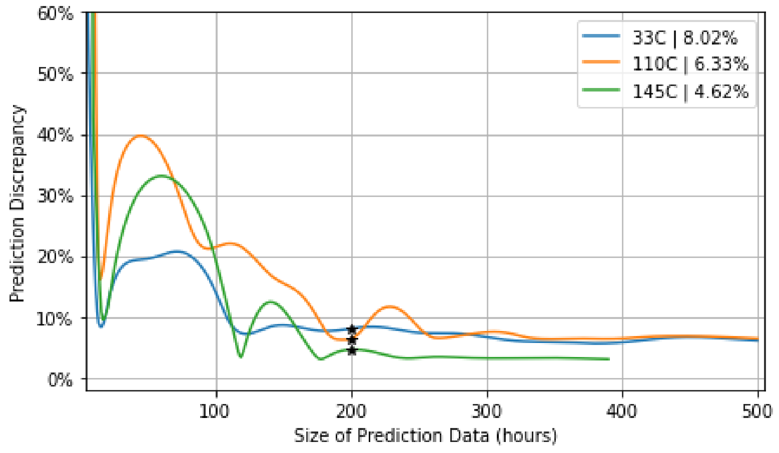


Figure 5.8: Variations in *Prediction Discrepancy* on increasing the size of the *Prediction Data*. Label contains the values of the highlighted points for *prediction data* set size of 200 hrs.

Table 5.2: Thermal Stress Experiments' Prediction Summary

Stress TEMP	**Stress Time**	**Prediction Data Size**	**Prediction Data Points**	**Prediction Discrepancy**		
				TRF	***dogbox***	***LM****
135 ^{o}C	1,000 hrs	400 hrs	400	2.05%	2.05%	40.2%$_{Failed}$
125 ^{o}C	1,000 hrs	400 hrs	400	2.76%	2.76%	32.9%$_{Failed}$
110 ^{o}C	500 hrs	200 hrs	200	6.33%	6.33%	6.33%
33 ^{o}C	500 hrs	200 hrs	200	8.02%	8.02%	8.02%
145 ^{o}C	390 hrs	200 hrs	200	4.62%	4.62%	4.62%

*Refer to Figure 5.4 to understand the failure.

5.2.5 Degradation Data Sampling Rate

In the semiconductor industry, it is inconvenient to sample the degradation data each hour. We learnt from an industrial client their preference to sample results less frequently than was our current sampling rate of one set of measurements per hour. Thus, the impact of the sampling rate on the *prediction discrepancy* was also analyzed. The results are shown in Figure 5.9. This figure is similar to Figure 5.6, although in Figure 5.9, each curve belongs to a different sampling rate (the sampling rate is mentioned

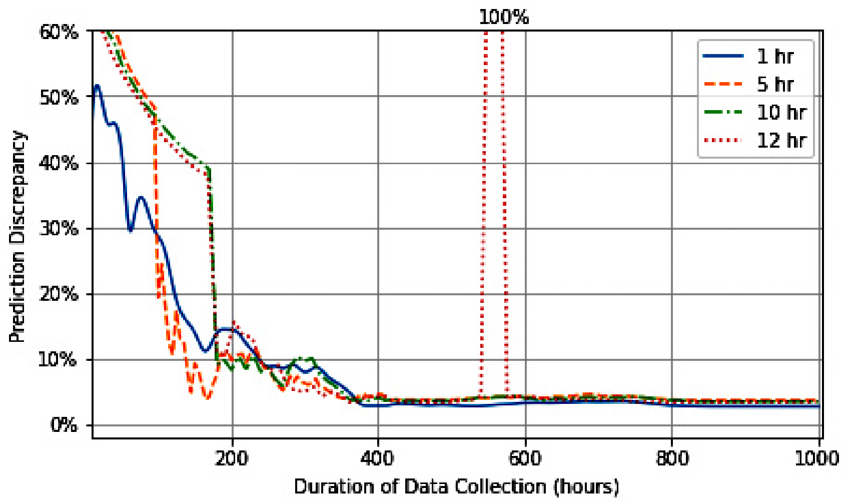


Figure 5.9: Variation in the *Prediction Discrepancy* curve on reducing the sampling rate from once per hour to once per 12 hrs.

in the heading). The solid curve illustrates when the sampling is performed each hour. The dashed curve (−−) reveals when the sampling was reduced to once every 5 hrs. The dotted curve (..) display when the sampling rate was further reduced to once every 12 hrs.

The analysis reveals that on reducing the sampling rate, the *prediction discrepancy* significantly changed if the duration of data collection were less than 200 hrs. However, post 200 hrs, the *prediction discrepancy* for different sampling rates remained fairly stable. Interestingly, past the ∼400 hrs of duration for the data collection, the *prediction discrepancy* saturated for all the sampling rates. This observation implies that our *prediction discrepancy* strongly depended on the duration of the data collection. This means that we needed the degradation data to form predictions (regardless of the sampling rate) until the half-time point of the experiment.

In the *12 hr* curve, for a very short time between 500 and 600 hrs, the prediction discrepancy shot to 100%. However, this was not observed in the higher sampling rate curves (*1 hr, 5 hr* and *10 hr*). For relatively lower sampling rates, the curve fitting function had fewer data points to perform the prediction. This made the curve fitting optimization function more sensitive to the noise in the data. Therefore, the sampling rate could not be reduced indefinitely.

5.3 Analysis II: Prediction Data Size vs Test Duration

This analysis is limited to Experiments #1 and #2 because they were the longest and their frequency degradation curves reveal a strict monotonic exponential decay profile. There are other reliability tests of shorter duration than HTOL; for instance, HAST at 110 ^{o}C lasts for ∼264 hrs and ELFR at 125 ^{o}C lasts for ∼168 hrs. To study the impact of this approach on such shorter reliability tests, we analyzed the requirements of the sizes of *prediction data* sets (durations of data collection) for different durations of the experiment.

During Analysis I, the **Stress Time**, i.e. the duration of experiment, remain fixed at 1,000 hrs. **Analysis I** showed that the degradation data from the initial hours of the experiment can indeed enable us to predict the degradation in the remainder of the experiment. At the finish of the Analysis I for the 1,000-hour long experiments, it was reported that ∼400 hrs of initial degradation data sufficed to achieve a discrepancy of less than 3%. If the duration of experiment were reduced, for instance, to 500 hrs, would we still need 400 hrs of degradation data, or would the degradation data collected for less than 400 hrs provide the same discrepancy? To explore this, the *prediction discrepancy* of 5% is considered an illustrative reference. We can compute the size of the *prediction data* set needed for every duration of the experiment to achieve a *prediction discrepancy* of ≤5%. It has been verified that the qualitative conclusion of this analysis will remain the same were one to choose 10%, rather than 5%, as the maximum tolerance level. Note that we were not required to conduct new experiments for comparative alternatives to every experimented duration to conduct this analysis. Instead, subsets of 1,000-hr long experiments' data were used.

The result of this computation for the experiments conducted at 135 ^{o}C and 125^{o}C, is shown in Figures 5.10 and 5.11, respectively. In both these figures, the x-coordinates of the curves represent the total durations of the experiments and the y-coordinates represents the required sizes of the *prediction data* sets (durations of data collection) to achieve the *prediction discrepancy* of ≤5%. It is evident from these figures, that the minimum required prediction data size was not a linear function of the duration of the experiment. Since the *actual* curves seemed to follow a logarithmic trend, we utilized a simple logarithmic function ($y = m\log(x) + c$) to quantify the non-linear relationship. Table 5.3 summarizes this observation by showing various points on the curves. By carefully examining the values on the table, it was found that, as the duration of experiment reduced, the required size of the *prediction data* set (in the column **Required Data** *(%)*) increased. This means that, for smaller duration experiments, we need to acquire a larger (by waiting longer) portion of the inquiry to make a prediction at the same level of confidence.

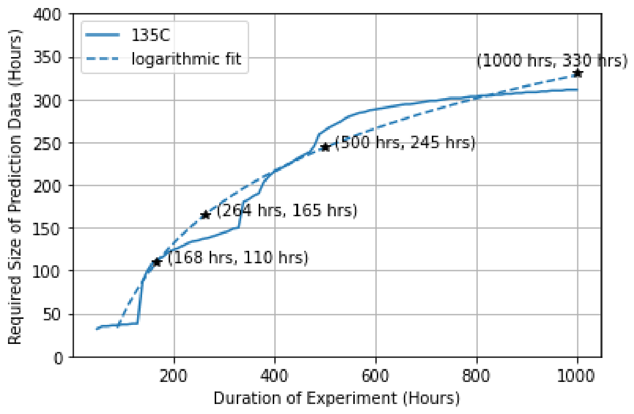


Figure 5.10: *Actual* (solid) and *logarithmic fit* ('−−') curves representing the variation in minimum required size of *Prediction Data* for different duration of experiments to reach a discrepancy of ≤5% at 135 ^{o}C.

Table 5.3: *Prediction Data* Size Required for ≤5% Discrepancy

Experiment Duration *(hrs)*	**135 ^{o}C**		**125 ^{o}C**	
	Required Data *(hrs)**	**Required Data** *(%)*	**Required Data** *(hrs)**	**Required Data** *(%)*
1,000	330	33%	365	37%
500	245	49%	270	54%
264**	165	63%	180	68%
168**	110	66%	117	70%

*Using the *logarithmic fit* equation.
**Points highlighted to illustrate the impact on HAST and ELFR reliability tests.

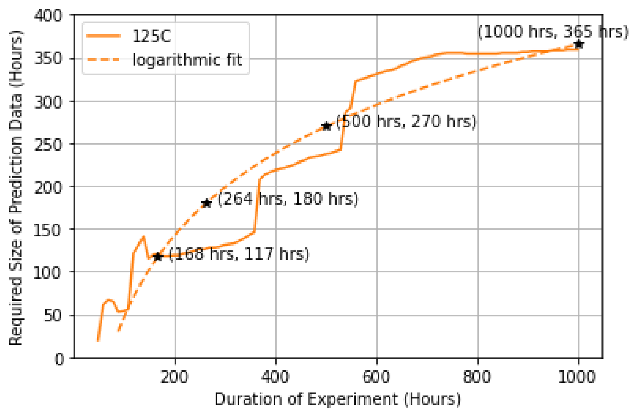


Figure 5.11: *Actual* (solid) and *logarithmic fit* ('−−') curves representing the variation in minimum required size of *Prediction Data* for different duration of experiments to reach a discrepancy of $\leq$5% at 125 ^{o}C.

5.4 Conclusion

This chapter describes multiple level of analyses on the frequency degradation data obtained from all the thermal stress experiments. From the observations and a systematic analysis, it was demonstrated that it is indeed possible to predict the circuit degradation in a relatively shorter time frame. This was found to be consistent across temperatures. Also, the discrepancy in the prediction was analyzed for different sizes of prediction data which showed there is an optimum time-spot during the accelerated aging beyond which any data captured is redundant for the purpose of prediction. After studying the impact of the approach to other reliability tests of shorter duration, it was observed that, as the duration of the experiment decreases, we require data for a larger portion of the experiment to make a prediction at a given level of confidence.

Chapter 6

Conclusion and Future Work

6.1 Answer to the Main Research Question

It will be recalled from the discussion in Chapter 1 that the motivation for this research was to assist the semiconductor industry in preventing any potential delays in their products' time-to-market. HTOL exemplifies a reliability test whose ambitious qualification criteria can at times add significant delay to the product's pre-market period. Thus, to prevent this, reducing the HTOL duration is of considerable importance. This defined the main objective of this research where the aim was to determine the feasibility of forecasting HTOL-like reliability test results from the degradation data collected during the initial duration of that test.

As we previously observed that at the die-level there are several aging mechanisms, such as BTI, HCI, TDDB and EM, which contribute to the aging of a semiconductor device, this research was started by first exploring the feasibility of the approach on a single mechanism, i.e. BTI, so that the observations could be conclusive. An RO-based architecture (Figure 3.1) was employed, which has BTI as the dominant aging mechanism in its given stress conditions. Several thermally accelerated aging tests were performed on it across diverse temperatures and their aging data were studied with the primary aim of forecasting their degree of aging from their test data collected for a relatively shorter time frame. From that analysis, it was demonstrated that the approach taken to associate a portion of accelerated aging data to the circuit's aging model can indeed accurately predict the accelerated aging of the circuit for the remaining hours until the end of the test.

Since a prediction is usually not 100% accurate, the error in the predictions for diverse sizes of *prediction data* set was also analyzed along with the redundancy from collecting degradation data beyond some time-point. This analysis qualitatively demonstrates the existence of an optimum time-spot below which we cannot reduce the test duration without incurring a penalty while attempting to forecast the degree of aging, and beyond which any additional data fails to provide significant improvement to our accuracy in regards to BTI-induced aging.

To expand the application of this approach to other reliability tests of shorter duration, including HAST and ELFR, the minimum size of aging data needed for any arbitrary test duration to forecast aging was also analyzed. It was observed that the size of *prediction data* relative to the test duration increases as the duration of the test reduces. In other words, for shorter accelerated aging tests, we are required to acquire data for a longer portion of the test to make a prediction at a given accuracy.

6.2 Summary of the Research Work

6.2.1 TYNQ Performance

Our reliability test platform TYNQ, established the foundation for our reliability experiments. Its *Self-heating Module* made it possible for it to partially emulate the HTOL test conditions. It showcased the viability of conducting high-temperature experiments without the use of large thermal chambers like thermostream or ovens. The software module of TYNQ, including TestChip Driver (*TestChip.py*), enabled various measurements on the chip, controlled the SHEs to regulate the die temperature, and regularly transmitted the data over the web for backup, without any intervention, thereby removing any dependency on any external laboratory users or equipment.

However, during the research, an important limitation to the *Self-heating Module* was encountered. The feedback for the **Temperature Regulation** feature arises from a single-fixed point on the die. Due to this, in reality, the temperature was being regulated at only a single point on the die. The feature does not take into consideration the placement/distribution of SHEs (Self-heating elements) on the die. Because of this, it fails to maintain the heat distribution on the die for many hours, which was evident in Experiment #3 conducted at 110 ^{o}C. However, this heat distribution problem could possibly be resolved were the **Temperature Regulation** feature made aware of the locations of all the SHEs, so that it could uniformly select those SHEs to be turned on. Thus, until the *Self-heating Module* was upgraded with such adjustments, the *Temperature Regulation* feature of the TYNQ would be limited establishing and regulating the temperature above a certain level.

For these experiments to run uninterrupted for **1,000 hrs**, automation proved to be of importance as it enabled us to anticipate necessary hardware and software components to minimize human intervention during the test, and provided a structure for the reliability experiments. Automation also facilitated replicating the experimental conditions. For the scientific community, proving the viability of automation in reliability experiments can aid in expediting the development time of test setups.

6.2.2 Experimental Result Adaptability

The fundamental nature of our approach made it feasible for predicting the results of a thermally accelerated aging test. The computation of the circuit-level aging model (Equation 5.6) is a crucial step in this process. But the circuit-level aging model is subjective. If the circuit changes, the circuit-level aging model should be modified accordingly. Thus, for any user to adopt the approach that has been described in this thesis, the primary requirement is a good estimation of a mathematical relationship between the circuit's reference parameter of degradation and the V_T of the transistor. It is this relationship which could be used to transfer the transistor-level to the circuit-level aging model.

An important limitation to the reported observations and conclusions is that they are limited to BTI-induced accelerated aging tests on MOSFETs. The reason for this is that the transistor-level aging model (Equation 5.5) describes only the BTI-induced V_T shift for MOSFETs. Therefore, a reliability test user intending to predict their circuit aging using this approach should ascertain that BTI is the dominant aging mechanism in their circuit during their accelerated aging test. However, in the literature, HCI and TDDB transistor-level aging models are available that can be adopted to explore this research avenue when applied to respective classes of circuits, which constitutes the possibilities for future work on this approach.

6.3 Recommendations for Future Work

So far in this research, the reliability of semiconductor devices is explored using temperature as the catalyst to accelerate the aging process, with a focus on verifying the approach with the BTI aging mechanism applied to

FPGAs. Henceforth, I see three potential research avenues which could follow:

1. **Voltage:** As voltage stress can also accelerate the aging process, similar research work with voltage as the aging accelerator could also be performed to further explore the viability of the approach for HTOL. The reason is that, in HTOL, aging acceleration is accomplished using temperature and voltage combined, to compound their effects. Therefore, exploring the voltage catalyst will take this approach a step closer to its applicability for predicting HTOL test results.

2. **HCI, TDDB, and EM:** The aging of a device in the field is the combined effect of all the aging mechanisms, and the same holds for HTOL. Therefore, it is important to explore the viability of the approach on other aging mechanisms. For that purpose, similar research work can be performed on circuits where other aging mechanisms dominate. X. Li *et al.* in their article [34] also describe HCI and TDDB transistor-level aging model which can be used to predict the accelerated aging induced by HCI and TDDB mechanisms.

3. **Custom ASIC:** This research work has been conducted on FPGAs where the user is constrained to implement circuits using LUTs. However, in custom designed ASICs (Application Specific Integrated Circuits) the user has transistor-level access to design, implement, and study the approach on specific circuits with their desired technology.

These three research avenues are not completely independent of each other. Depending upon the accessibility to researcher(s), either one or, simultaneously, all these avenues can be explored. An example might involve designing a custom ASIC chip to study the aging acceleration due to voltage on a circuit with HCI as the dominant mechanism. Table 6.1 shows where this approach currently stands. I speculate that when all the categories are checked, this approach of forecasting accelerated aging test results will have achieved the level by which it might be used on a regular basis to predict industry-standard HTOL test results.

Table 6.1: Possible Future Research Avenues

Work	**Aging Accelerator**		**Aging Mechanism**				**Test Chip**
	Temperature	**Voltage**	**BTI**	**HCI**	**TDDB**	**EM**	
This research project [35, 36]	✓		✓				FPGA

Bibliography

[1] S. A. Sheikholeslam, P. Chanawala, P. Palanichamy, and A. Ivanov, “Delay analysis for fpga devices operating at unconventional temperatures,” 2021. [Online]. Available: https://github.com/sarashs/TYNQ

[2] XILINX, “Vivado design suite 7 series fpga and zynq-7000 soc libraries guide,” 2021. [Online]. Available: https://docs.xilinx.com/v/u/2019.1-English/ug953-vivado-7series-libraries

[3] JEDEC, “Jedec standards and documents,” 1958. [Online]. Available: https://www.jedec.org/standards-documents

[4] I. Hill, P. Chanawala, R. Singh, S. A. Sheikholeslam, and A. Ivanov, “Cmos reliability from past to future: A survey of requirements, trends, and prediction methods,” *IEEE Transactions on Device and Materials Reliability*, vol. 22, no. 1, pp. 1–18, 2022.

[5] A. Lawrence and J. VerWey, “The automotive semiconductor market – key determinants of u.s. firm competitiveness,” 2019. [Online]. Available: https://www.usitc.gov/publications/332/executive_briefings/ebot_amanda_lawrence_john_verwey_the_automotive_semiconductor_market_pdf.pdf

[6] V. Gahir, “Insights on automotive design with veejay gahir,” 2014. [Online]. Available: https://www.lynda.com/Automotive-Design-tutorials/Insights-Automotive-Design-Veejay-Gahir/163418-2.html

[7] S. Agarwal, “Mil/jedec standards update,” 2016. [Online]. Available: https://nepp.nasa.gov/workshops/etw2016/talks/13MON/20160613-1000-Agarwal-2016%20NASA%20ETW_13-16June%202016_MIL%20JEDEC%20Standards%209Jun2016.pdf

[8] L. Condra, D. Followell, G. Houchens, J. Jenks, M. Koehler, and Z. Porter, “Minimizing the effects of electronic component

obsolescence," 2000. [Online]. Available: http://www.boeing.com/commercial/aeromagazine/aero_10/elect_textonly.html

[9] B. E. A. Group, "Life span of biomedical devices," 2004. [Online]. Available: http://cedglobal.org/download/Life%20Span%20of%20Biomedical%20Devices%20-%20Guidance%20Paper%20Final.pdf

[10] JC-14, "Temperature, bias, and operating life," 2017. [Online]. Available: https://www.jedec.org/sites/default/files/docs/22A108F.pdf

[11] ——, "Early life failure rate calculation procedure for semiconductor components." [Online]. Available: https://www.jedec.org/system/files/docs/JESD74A_RA.pdf

[12] ——, "High temperature storage life." [Online]. Available: https://www.jedec.org/sites/default/files/docs/22a103D.pdf

[13] J. W. McPherson, "Brief history of jedec qualification standards for silicon technology and their applicability to wbg semiconductors," in *2018 IEEE International Reliability Physics Symposium (IRPS)*, 2018, pp. 3B.1–1–3B.1–8.

[14] S. W. Pae, H. C. Sagong, C. Liu, M. J. Jin, Y. H. Kim, S. J. Choo, J. J. Kim, H. J. Kim, S. Y. Yoon, H. W. Nam, H. W. Shim, S. M. Park, J. K. Park, S. C. Shin, and J. W. Park, "Considering physical mechanisms and geometry dependencies in 14nm finfet circuit aging and product validations," in *2015 IEEE International Electron Devices Meeting (IEDM)*, 2015, pp. 20.6.1–20.6.4.

[15] R. Kwasnick, M. Reilly, J. Hatfield, S. C. Johnson, and A. Rahman, "Impact of vlsi technology scaling on htol," in *2012 IEEE International Reliability Physics Symposium (IRPS)*, 2012, pp. 5C.3.1–5C.3.5.

[16] L. Yu, J. Ren, X. Lu, and X. Wang, "Nbti and hci aging prediction and reliability screening during production test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 3000–3011, 2020.

[17] W.-T. K. Chien, Y. A. Zhao, Y. Zhu, and Y. Song, "Early detection and prediction of hkmg sram htol performance by wlr pbti tests," *Microelectronics Reliability*, vol. 64, pp. 185–188, 2016, proceedings of the 27th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[18] X. Guo and M. R. Stan, *Accelerated Active Self-Healing for Integrated Circuits*. Cham: Springer International Publishing, 2020, pp. XIX–208. [Online]. Available: https://doi.org/10.1007/978-3-030-20051-0

[19] I. Stratakos, K. Maragos, G. Lentaris, D. Soudris, and K. Siozios, *Aging Evaluation and Mitigation Techniques Targeting FPGA Devices*. CRC Press, 12 2018, pp. 131–149.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] S. Mahapatra, V. Huard, A. Kerber, V. Reddy, S. Kalpat, and A. Haggag, "Universality of nbti - from devices to circuits and products," in *2014 IEEE International Reliability Physics Symposium*, 2014, pp. 3B.1.1–3B.1.8.

[25] X. Guo and M. R. Stan, *Accelerated and Active Self-healing Techniques for BTI Wearout*. Cham: Springer International Publishing, 2020, pp. 17–55. [Online]. Available: https://doi.org/10.1007/978-3-030-20051-0_2

[26] ——, *Design and Aging Challenges in FinFET Circuits and Internet of Things (IoT) Applications*. Cham: Springer International Publishing, 2020, pp. 143–189. [Online]. Available: https://doi.org/10.1007/978-3-030-20051-0_6

[27] ——, *Accelerating and Activating Recovery Against EM Wearout*. Cham: Springer International Publishing, 2020, pp. 57–75. [Online]. Available: https://doi.org/10.1007/978-3-030-20051-0_3

[28] DIGILENT, “Pynq-z1 board reference manual,” 2017. [Online]. Available: https://digilent.com/reference/_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf

[29] XILINX, “Zynq-7000 soc data sheet: Overview,” 2018. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview

[30] M. Naouss and F. Marc, “Fpga lut delay degradation due to hci: Experiment and simulation results,” *Microelectronics Reliability*, vol. 64, pp. 31–35, 2016, proceedings of the 27th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271416301925

[31] 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.

[32] A. Amouri, J. Hepp, and M. Tahoori, “Self-heating thermal-aware testing of fpgas,” in *2014 IEEE 32nd VLSI Test Symposium (VTS)*, 2014, pp. 1–6.

[33] ——, “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.

[34] 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.

[35] P. Chanawala, I. Hill, S. A. Sheikholeslam, and A. Ivanov, “Prediction of thermally accelerated aging process at 28nm,” in *2022 IEEE European Test Symposium (ETS)*, in press.

[36] ——, “Forecasting reliability test result from partial data for bti-induced aging,” *IEEE Electron Device Letters*, to be submitted.

Appendix A

6-input Look-up Table with 1 Output

Verilog code used for instantiating a *NOT* gate in the RO-architecture of Figure 2.3 and 3.1. The *NOT* gate is built on a 6-input 1-output look-up table on Zynq XC7Z020-1CLG400C FPGA.

```
module LUT6_NOT(
    input in_sig,
    output out_sig
    );
// This is important to make the inverters consistent
(* LOCK_PINS="I0:A6, I1:A1, I2:A2, I3:A3, I4:A4, I5:A5" *)
LUT6 #(.INIT(64'h5555555555555555)) // This should make an
    inverter
LUT6_inst (
 .O(out_sig), // LUT general output
 .I0(in_sig), // LUT input
 .I1(1'b0), // LUT input
 .I2(1'b0), // LUT input
 .I3(1'b0), // LUT input
 .I4(1'b0), // LUT input
 .I5(1'b0) // LUT input
);
endmodule
```

Appendix B

6-input Look-up Table with 2 Outputs

Verilog code used for instantiating a *one2two* gate in the RO-architecture of Figure 2.3. The gate is built on a 6-input 2-output look-up table on Zynq XC7Z020-1CLG400C FPGA.

```verilog
`timescale 1ns / 1ps

module one2two(
(*DONT_TOUCH= "true"*)     input input1,
 (*DONT_TOUCH= "true"*)    output output1,
 (*DONT_TOUCH= "true"*)    output output2
    );
(* LOCK_PINS="I0:A1, I1:A2, I2:A6, I3:A3, I4:A4, I5:A5" *)
LUT6_2 #(.INIT(64'hAAAAAAAAAAAAAAAA)) // Specify LUT Contents
LUT6_2_inst (
.O6(output1), // 1-bit LUT6 output
.O5(output2), // 1-bit lower LUT5 output
.I0(input1), // 1-bit LUT input
.I1(1'b0), // 1-bit LUT input
.I2(1'b0), // 1-bit LUT input
.I3(1'b0), // 1-bit LUT input
.I4(1'b0), // 1-bit LUT input
.I5(1'b0) // 1-bit LUT input
);
endmodule
```

Appendix C

Frequency Counter

Verilog code used for instantiating a *frequency counter* for the RO-architecture of Figure 2.3 and 3.1.

```
`timescale 1ns / 1ps

module frequency_counter #
    (
    parameter num_counters = 4
    )
  (
    input wire [num_counters-1:0] in_signal,
    input clk,
    output [num_counters*32 - 1:0] freq
    );

reg [num_counters*32 - 1 :0] freq_count;
reg [31:0] clk_count;
reg clk_done;
reg [num_counters*32 - 1:0] freq_out;


(*DONT_TOUCH= "true"*) assign freq = freq_out;

generate
genvar i;
for (i = 0; i < num_counters; i = i+1) begin

always @(posedge in_signal[i]) begin
    (*DONT_TOUCH= "true"*) freq_count[(i+1)*32 - 1:i*32] =
   freq_count[(i+1)*32 - 1:i*32] + 1;
    if (clk_done == 1) begin
        freq_count[(i+1)*32 - 1:i*32] = 0;
    end
end

end
endgenerate

always @(posedge clk) begin
  clk_count <= clk_count + 1;
```

```verilog
    clk_done = 1'b0;
  if ((clk_count > 99998) & (clk_count < 100010)) begin //we
    stop the count for a full milisecon
      clk_done = 1'b1;
  end
  else if (clk_count == 100010) begin
    clk_count <= 0;
    clk_done = 1'b0;
  end
end


always @(posedge clk_done) begin
   freq_out = freq_count;
end

endmodule
```

Appendix D

TYNQ TestChip Driver

TestChip.py [1]: TestChip driver is a 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 the 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.

Appendix E

CUT's MUX stage

Verilog code used for instantiating a *MUX* gate in the RO-architecture of Figure 3.1. The gate is built on a 6-input 2-output look-up table on Zynq XC7Z020-1CLG400C FPGA.

```
`timescale 1ns / 1ps

module LUT6_FB_RO(
    (*DONT_TOUCH= "true"*) input En,
    (*DONT_TOUCH= "true"*) input r_out,
    (*DONT_TOUCH= "true"*) input f_i,
    (*DONT_TOUCH= "true"*) output r_in,
    (*DONT_TOUCH= "true"*) output freq_count
    );
LUT6_2 #(.INIT(64'hCACACACACACACACA))
LUT6_2_inst (
.O6(r_in), // 1-bit LUT6 output
.O5(freq_count), // 1-bit lower LUT5 output
.I0(f_i), // 1-bit LUT input
.I1(r_out), // 1-bit LUT input
.I2(En), // 1-bit LUT input
.I3(1'b0), // 1-bit LUT input
.I4(1'b0), // 1-bit LUT input
.I5(1'b0) // 1-bit LUT input
);
endmodule
```

Appendix F

Temperature Regulation

Python source code of *fix_temperature* function. It is embedded inside the *experiment_flow.py* script. The function regulated the chip temperature during the thermal stress experiments.

```python
def fix_temperature(self, desired_temperature):
        """simple control scheme to fix the temperature to a desired value

        Returns: None
        """

        if self.XADC_temp() > (desired_temperature +
                               self.temp_ctrl_sensitivity):
            self.temp_ctrl_intensity -= 1
            self.heat_on(self.temp_ctrl_intensity)
        elif self.XADC_temp() < (desired_temperature -
                                 self.temp_ctrl_sensitivity):
            self.temp_ctrl_intensity += 1
            self.heat_on(self.temp_ctrl_intensity)
        if self.temp_ctrl_intensity > self.max_intensity:
            self.temp_ctrl_intensity = self.max_intensity
        elif self.temp_ctrl_intensity < 0:
            self.temp_ctrl_intensity = 0
```

Appendix G

Read CUT Frequency

Python source code of *read_multi_ro* function. It is embedded inside the *experiment_flow.py* script. The function captured the frequency counter output of mutiple ROs during the experiment.

```python
def read_multi_RO(self, RO_dict):
        """Reads the frequency of selected ROs
        Parameters:
        RO_dict : {keys=RO_ip_name, values=[RO list per IP]}

        Returns: freq_dict {keys=RO_ip_name, values=[
    frequencies (nparray)]}
        """
        freq_dict = {}
        for item in RO_dict.keys():
            RO_list = RO_dict[item]
            len_ro = len(RO_list)
            freq_list = np.zeros((len_ro))
            RO = getattr(self, item)
            for i in range(len_ro):
                freq_list[i] = RO.read(
                    self.RO_base_address +
                    RO_list[i] * self.
    counter_address_increament
                )/1000
            freq_dict[item] = freq_list
        return freq_dict
```

Appendix H

Data Capture and Storage

Python source code of *record* function. It is embedded inside the *experiment_flow.py* script. The function assembled all the measured values, including temperature, voltage and oscillation frequency of the CUTs.

```python
def record(ol, total_duration, every, num_oscillators, desired_temp):
    """
    parameters
    ----------
    ol               : FPGA bitstream file location
    total_duration   : total duration in seconds
    every            : sampling step size in seconds
    num_oscillators  : number of CUTs per instantiation of CUT Module
    desired_temp     : desired stress temperature
    """
    data = []
    RO_dict = dict(RO2_0=list(range(num_oscillators)))
    times = []
    init_time = datetime.now()
    now_time = init_time
    while(now_time < (init_time + total_duration)):
        ol.fix_temperature(desired_temp)
        now_time = datetime.now()

        temperature = ol.XADC_temp() # Read Temperature

        vbram = ol.XADC_voltage('vbram') # Read Voltages

        output_dict = ol.read_multi_RO(RO_dict) # Read freq.

        # Stack all the measurements
        current_read = np.hstack(output_dict['RO2_0'])
        current_read = np.hstack((current_read, np.array([temperature])))
        current_read = np.hstack((current_read, np.array([vbram])))

        now_pacific_live = datetime.now(timezone('US/Pacific'))
        times.append(now_pacific_live)

        data.append(current_read)

        while(datetime.now() < now_time + every):
            if(every > timedelta(seconds=1)):
                sleep(1)
                ol.fix_temperature(desired_temp)
            pass

    data = np.vstack(data)

    output = pd.DataFrame(data, columns=([f'RO2_0{i}' for i in
    range(num_oscillators)] + ['Temperature'] + ['vbram']))

    output['Timestamp'] = pd.DataFrame(dict(Timestamp=times))

    return output
```

Appendix I

Dispatch Email

Python source code of *sendemail* function. It is embedded inside the *experiment_flow.py* script. During the experiments, it hourly emailed the attachments that contained all the measured values.

```python
def sendemail(self, subject, filename, receiver_email):
        body = "This email is sent from Pynq board"
        sender_email = "###@gmail.com" #Mention the email id
        password = "###" #Mention that id's password

        # Create a multipart message and set headers
        message = MIMEMultipart()
        message["From"] = sender_email
        message["To"] = receiver_email
        message["Subject"] = subject

        # Add body to email
        message.attach(MIMEText(body, "plain"))

        # Open file in binary mode
        with open(filename, "rb") as attachment:
            part = MIMEBase("application", "octet-stream")
            part.set_payload(attachment.read())

        # Encode file in ASCII characters to send by email
        encoders.encode_base64(part)

        # Add header as key/value pair to attachment part
        part.add_header(
            "Content-Disposition",
            f"attachment; filename= {filename}",
        )

        # Add attachment to message & convert message to string
        message.attach(part)
        text = message.as_string()

        # Log in to server using secure context and send email
        try:
            context = ssl.create_default_context()
            with smtplib.SMTP_SSL("smtp.gmail.com", 465,
    context=context) as server:
                server.login(sender_email, password)
                server.sendmail(sender_email, receiver_email,
    text)
        except:
            pass
```

Appendix J

Frequency vs. Temperature Phase

Python source code of the **Frequency vs Temperature** phase of the experiment. Its description can be found in Section 3.4.

```python
ol = TestChip(f'/home/xilinx/pynq/overlays/###/###.bit') # Mention the location of the FPGA bitstream
ol.en_feedback(1) #Setting Mode to 1
for k in range(50):
    sleep(2)
    ol.fix_temperature(T_desired - 5)

temp_data = np.arange(T_desired - 5,T_desired + 6,2);
for j in range(temp_data.shape[0]):

    for k in range(20):
        sleep(1)
        ol.fix_temperature(temp_data[j])

    output = record(ol, pre_total_duration, pre_every, 20, temp_data[j])
    output.to_pickle(f'./###{j}.pkl') #Name your pickle file

    #send email
    filename = f'###{j}.pkl' #Mention your pickle filename
    subject = f'###{j}' #Give a unique subject to your email
    receiver_email = '###@gmail.com' #Mention the receiver's email address
    ol.sendemail(subject,filename,receiver_email)
```

Appendix K

Stress and Measurement Phase

Python source code of the **Stress and Measure** phase of the experiment. Its description can be found in Section 3.4.

```python
for i in range(1001):
    ol.en_feedback(0)
    output = record(ol, stress_total_duration, sampling_rate,
   num_oscillators, T_desired)
    output.to_pickle(f'./###{i}.pkl')  #Name your pickle file

    ol.en_feedback(1)
    output = record(ol, measure_total_duration, sampling_rate,
   num_oscillator, T_desired)
    output.to_pickle(f'./###{i}.pkl')  #Name your pickle file

    # send email
    filename = f'###{i}.pkl' #Mention your pickle filename
    subject = f'[###{i}' #Give a unique subject to your email
    receiver_email = '###@gmail.com' #Mention the receiver's
   email address
    ol.sendemail(subject, filename, receiver_email)

    filename = f'###{i}.pkl' #Mention your pickle filename
    subject = f'[###{i}' #Give a unique subject to your email
    receiver_email = '###5@gmail.com' #Mention the receiver's
   email address
    ol.sendemail(subject, filename, receiver_email)
```

Appendix L

Clock Divider

Verilog code used for instantiating a clock divider using D Flip-Flops.

```
`timescale 1ns / 1ps

module clk_div(
    input wire clk_in,
    output wire clk_1,  //100MHz
  output wire clk_5,  //6.25MHz
  output wire clk_9,  //390KHz
  output wire clk_13, //24KHz
  output wire clk_14  //12KHz
    );

wire [13:0] din;
wire [13:0] clkdiv;

dff dff_inst0 (
.clk(clk_in),
.rst(0),
.D(din[0]),
.Q(clkdiv[0])
);

genvar i;
generate
for (i = 1; i < 14; i=i+1)
begin : dff_gen_label
  dff dff_inst (
    .clk(clkdiv[i-1]),
    .rst(0),
    .D(din[i]),
    .Q(clkdiv[i])
  );
  end
endgenerate

assign din = ~clkdiv;

assign clk_1 = clkdiv[0];
assign clk_5 = clkdiv[4];
```

```verilog
assign clk_9 = clkdiv[8];
assign clk_13 = clkdiv[12];
assign clk_14 = clkdiv[13];

endmodule
```