Universiteit Leiden
The Netherlands

Opleiding Informatica

On The Vulnerabilities of FPGAs to Power Hammering Circuits

Lennart van Drunick

Supervisors:
Dr. Todor P. Stefanov & Abolfazl Sajadi

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/11/2025

Abstract

Since 1985, Field Programmable Gate Arrays (FPGAs) have been used across various electronic systems due to their flexibility, performance, and cost-effectiveness. However, this flexibility also introduces specific security risks. One such threat is the power hammering loop, a malicious circuit designed to draw excessive power, potentially causing system instability or damage. Today, even open-source Hardware Description Language (HDL) designs for FPGAs may unknowingly include hidden loops that are activated under certain conditions, posing risks in security-critical applications. This thesis examines the vulnerability of the Artix-7 FPGA (XC7A100T-2FTG256) to power hammering circuits and evaluates whether recent versions of the Vivado design tool chain can detect such circuits. Custom power hammering loops are developed, simulated in Vivado, and implemented on a CW305 development board. Real-world power data is collected using the Nordic Power Profiler Kit to assess detection effectiveness and potential risks in current FPGA workflows.

Contents

1 Introduction

With the introduction of the XC2064 chip in 1985, shown in Figure 1 [IEE17], Field Programmable Gate Arrays (FPGAs) have become increasingly popular across a wide range of digital systems due to their flexibility, high performance, and cost-effectiveness. Their reconfigurability allows developers to tailor hardware behavior for specific applications, making them a valuable tool in areas like embedded systems, networking equipment and cryptographic modules.
However, this same flexibility introduces unique security challenges. Among them is the risk posed by *power hammering circuits*, malicious circuit structures designed to consume excessive power and potentially cause system instability or physical damage. Such circuits can be cleverly hidden in legitimate-looking Hardware Description Language (HDL) code and triggered under specific conditions.

Figure 1: *First Xilinx FPGA: XC2064*

For example, a developer might unknowingly deploy open-source HDL code into a security-critical system without realizing it contains embedded power hammering circuits. These circuits may be activated after a delay or in response to a specific signal, thereby disabling the system or reducing its operational lifespan through overheating or power disruption. This thesis investigates both the technical feasibility of these circuits on a modern mid-range FPGA, and the effectiveness of the Vivado development toolchain in identifying and mitigating them.

To make this investigation concrete and manageable, the scope is limited to the Artix-7 FPGA (XC7A100T-2FTG256), mounted on a CW305 development board [Mou25]. This FPGA is widely used, well-documented, and cost-effective, making it an ideal candidate for exploring real-world attack feasibility. Power consumption measurements are performed using the Nordic Power Profiler Kit (PPK2) alongside Vivado simulations to estimate the circuits switching activity and operating frequency. The goal is to determine whether malicious circuits can still bypass the Vivado detection mechanisms and operate without triggering clear warnings.

1.1 Research Questions

While the main research question is embedded in the thesis title, it can be formally and specifically restated as:

> **How vulnerable is the Artix-7 FPGA (XC7A100T-2FTG256) to power hammering circuits, and how effectively can recent versions of the Vivado design tool chain detect these circuits?**

This research builds upon prior studies such as *FPGADefender* [LMG+20], which shows that earlier versions of Vivado failed to detect some power hammering circuits, potentially enabling silent hardware disruption. Here, we reevaluate these findings using more recent versions of Vivado [AMD25] and extend them by re-implementing and measuring the impact of selected circuits on physical hardware. To this end, it is necessary to divide the investigation into three sub-questions.

The first step in assessing the FPGA vulnerability is understanding the baseline behavior of the FPGA. Power hammering circuits are designed to draw an excessive amount of power, but to measure this increase meaningfully, a reference point is required. Without knowledge of the FPGAs typical power consumption during idle or normal operations, we cannot quantify the impact of the injected malicious designs. This leads us to the first sub-question:

> **What is the normal power usage of the Artix-7 FPGA, and how does it change when power hammering circuits are introduced?**

Once the increase in power consumption is measured, the next step is to evaluate its consequences. High power consumption may not always result in visible damage or system failure, depending on the tolerance of the device and the characteristics of the power hammering circuits. To fully understand the threat these circuits pose, it is essential to test how they affect the system stability under different conditions. This raises the second sub-question:

> **Under what conditions do power hammering circuits cause instability, malfunction, or complete system failure?**

Finally, one of the main goals of this thesis is to determine whether current FPGA development tools can detect these malicious circuits before they are deployed. While previous studies have shown that earlier versions of Vivado tools were not always capable of flagging harmful circuits, improvements may have been made in newer releases. Therefore, the third sub-question is:

> **Can recent versions of Vivado detect these harmful circuits, either during circuit synthesis, implementation, or static analysis?**

When we combine these sub-questions they will act as a guide to the experimental methodology of this thesis, from power measurement and system observation to evaluating Vivado's tools response.

1.2 Contributions to the Field of Security

This thesis contributes to the field of FPGA security by providing both experimental insights and practical evaluations in the context of power hammering attacks. The main contributions of this work are as follows:

- Systematic measurments and quantification of the power consumption impact of several custom-designed power hammering circuits on a widely used mid-range FPGA, namely the Artix-7 (XC7A100T-2FTG256), providing a reference baseline for future studies.
- Evaluations of the way power hammering circuits affect the system stability in real-world hardware conditions, identifying the threshold at which these circuits may cause malfunction or hardware disruption.
- Assessment of the effectiveness of the recent version of the Vivado 2042.2 design tools in detecting harmful circuit structures, using synthesis, implementation, and static analysis outputs.
- Revisiting of prior findings from related works such as *FPGADefender* [LMG+20] using updated toolchain versions and hardware, providing insights whether or not detection capabilities have improved over time.
- Offering a reproducible methodology and experimental setup, including the use of the CW305 board and PPK2, that future researchers can adopt to extend or validate their results.

Together, these contributions highlight the practical risks posed by power hammering circuits and assess the resilience of widely used FPGA development tools, informing both future research and security practices in FPGA-based system design.

1.3 Summary of the Thesis

This thesis is structured to systematically investigate the vulnerability of the Artix-7 FPGA to power hammering circuits, and to evaluate the detection capabilities of recent versions of the Vivado design suite. It begins with the introduction of key concepts and related work, continues with the experimental design and implementation, and concludes with the measurements, analysis, and final conclusions.

Chapter 2 defines the terminology used throughout the research and introduces the hardware and software tools employed in the design and implementation of the circuits, including the Artix-7 FPGA, CW305 board, and the Vivado toolchain.

Chapter 3 discusses previous research that serves as the basis of this study, with a particular focus on power hammering circuits and their interaction with FPGA development tools. It also examines the capabilities and limitations of the Vivado software in detecting potentially harmful circuits.

Chapter 4 describes the hardware and software setup, as well as the methodology used for circuit design, simulation, and implementation. This chapter outlines the development flow and details the practical steps taken to prepare the circuits for evaluation.

Chapter 5 presents the results and observations obtained by both software simulations and physical FPGA measurements.

Chapter 6 concludes the thesis by summarizing the key findings, interpreting their implications, and discussing the broader relevance of the results. It also reflects on the limitations of the study and proposes directions for future work to build upon these findings.

This bachelor thesis has been conducted at the Leiden Institute of Advanced Computer Science (LIACS), under the supervision of Dr. Todor P. Stefanov and Abolfazl Sajadi. It aims to contribute to the growing field of FPGA security by assessing practical risks and toolchain resilience in real-world development environments.

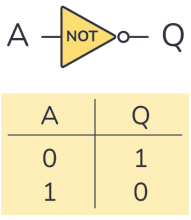
2 Background

With attacks targeting hardware resources becoming more sophisticated, it is critical to examine the architectural features that make FPGAs both powerful and potentially vulnerable. One such vulnerability is the presence of *power hammering circuits*, malicious hardware configurations that intentionally consume excessive power and can compromise system integrity. Before exploring the specific mechanics of these circuits, this chapter begins by establishing a basic understanding of the FPGA fabric. The following section introduces the core components and terminology necessary to contextualize how these attacks are implemented, detected, or overlooked during the design process.

2.1 FPGA Terminology

To fully understand how various power hammering circuits function on an FPGA, it is essential to first define the key terminology and components involved. FPGAs are highly flexible and reprogrammable integrated circuits that can be reconfigured to perform a wide range of digital logic functions. The name originates from the underlying architecture: a grid, or array, of logic gates with interconnects that can be programmed to implement custom hardware circuits. This reconfigurability makes FPGAs extremely powerful for applications ranging from digital signal processing to embedded control systems. However, the same flexibility that enables highly optimized designs also introduces the risk of inefficient or even malicious configurations, such as power hammering circuits. These circuits are intentionally designed to draw excessive power and place significant stress on the device. FPGAs are composed of a variety of configurable logic elements. The most relevant components for understanding the power hammering behavior are presented below:

Look-Up Tables (LUTs): A LUT is a small memory block used to implement logic functions. In the Artix-7 FPGA, a typical 6-input LUT can be programmed to represent any Boolean function of up to six variables. For example, a LUT can be configured to behave as a NOT gate which functions like shown in Figure 2. When one input is active and the rest are grounded, the output is the logical inverse of the input, demonstrating the flexibility of a LUT configuration.



A	Q
0	1
1	0

Figure 2: *Truth table of a NOT gate*

Flip-Flops and Latches: These elements are used to store binary values and are essential for implementing sequential circuits. Flip-flops update their output on the rising or falling edge of a clock signal, whereas latches are level-sensitive and respond to signal levels rather than edges. In the context of this research, these elements play a crucial role in forming feedback loops that are often exploited in power hammering designs.

Multiplexers (MUXes): A multiplexer selects one of several input signals based on a control signal and forwards the selected input to the output. For example, a 2-to-1 MUX outputs either input 0 or input 1 depending on whether the select line is low or high. MUXes are instrumental in building conditional or dynamic circuits and can be manipulated to create loop structures in malicious circuits.

Adders: Adders perform arithmetic operations, typically binary addition. In the context of power hammering, the interest lies in how adders handle carry propagation. When multiple adders are chained or used in such a way that carry bits toggle frequently, the circuit switching activity increases significantly, resulting in elevated dynamic power consumption. This behavior can be exploited to stress the power delivery network of the FPGA.

Standard FPGAs vs. Embedded FPGAs (eFPGAs)
In addition to standard FPGAs, which are standalone programmable logic devices, another progressively relevant category is the *embedded FPGA (eFPGA)* [BGG24]. eFPGAs are reconfigurable FPGA cores integrated directly into a larger *System-on-Chip (SoC)*. This integration combines the hardware adaptability of FPGAs with the efficiency and specialization of ASIC components. eFPGAs offer several practical benefits: (I) they retain the reconfigurability of traditional FPGAs while gaining the performance, area, and power efficiency of SoC-level integration; (II) they allow post-manufacture hardware updates, which can be critical for correcting logic bugs or implementing new features; and (III) they are increasingly used in domains such as cryptographic modules, secure boot systems, AI accelerators, and wireless baseband processors, where hardware-level security is critical.
However, this tight integration also raises the stakes of a successful attack. Malicious hardware structures like power hammering circuits, if deployed in an eFPGA region, may have broader consequences, potentially destabilizing not just the reconfigurable portion, but the entire SoC, including fixed-function digital logic. While this thesis limits its scope to a standalone Artix-7 FPGA (XC7A100T-2FTG256) on a CW305 development board, the findings may generalize to eFPGA contexts in two key ways:

1. **Design Tool Vulnerabilities** – If Vivado (or comparable Electronic Design Automation tools) fails to detect power hammering circuits in standard FPGAs, similar vulnerabilities may exist when generating bitstreams for eFPGA fabrics.

2. **Power Integrity Risk** – Because power delivery networks in SoCs are typically shared across functional blocks, localized power stress in the eFPGA region could affect system-wide stability, especially in battery-constrained or thermally sensitive applications.

While experiments are physically performed on dedicated hardware, their implications reach beyond, potentially stimulating security assessments and verification strategies in emerging eFPGA-based designs.
Due to their integration with the rest of the system, attacks on eFPGAs may have broader consequences, potentially impacting the entire SoC. While this thesis focuses on a standalone Artix-7 FPGA, the findings may be relevant to eFPGA environments as well, particularly regarding the potential detection limitations in design tools like Vivado.

2.2 Power Hammering Circuits

Power hammering circuits are a class of malicious circuits designed to exploit the power consumption characteristics of FPGAs by continuously toggling logic elements at high frequencies. Such circuits are commonly constructed as ring oscillators, configured to maximize the circuits' switching activity, and therefore the power consumption, without performing any meaningful computation.

As illustrated in Figure 3, the simplest form of a ring oscillator consists of an *odd number of inverters* connected in series, where the output of the last inverter is fed back into the input of the first. This creates a closed feedback loop in which the signal continuously oscillates between high and low states. In the example shown in Figure 3, a 3-stage inverter loop forms such a circuit, producing a periodic waveform without the need for an external clock.

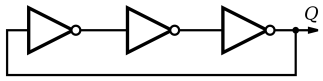


Figure 3: *A 3-stage ring oscillator composed of inverters*

Ring oscillators like these have legitimate applications in hardware design. They are often used to generate on-chip clock signals that are independent of the main system clock, or as components in true random number generators (TRNGs) [SNM20], due to their inherent jitter and unpredictability. In power hammering attacks, however, this concept is exploited maliciously. Rather than instantiating one or a few ring oscillators, an attacker may deploy thousands across the FPGA fabric. These ring oscillators operate in parallel, leading to extreme circuit switching activity across logic and routing resources. The result is a rapid increase in dynamic power consumption, which can overwhelm the power delivery system, elevate the on-chip temperature, degrade performance, or in severe cases, cause irreversible physical damage to the FPGA. As these circuits are compact and perform no functional computation, they can be hidden inside larger hardware designs. This makes them especially dangerous in open-source environments, where third-party HDL code may be integrated without sufficient review, potentially inserting hidden power hammering circuits that pass through standard design rule checks undetected.

2.3 Tools and Hardware

To design, simulate, and implement circuits on FPGAs, this research relies primarily on the *Vivado Design Suite*. Vivado is the industry-standard software tool for developing digital logic on Xilinx FPGAs. It allows users to describe hardware circuits using HDLs, simulate circuits behavior, estimate resource usage and power consumption, and generate configuration files for the FPGA. The circuits in this thesis are written in *Verilog*, a hardware description language widely used for modeling, designing, and implementing digital systems [Ver24]. Once the design is described, Vivado synthesizes it into a netlist composed of basic logic elements, such as LUTs, flip-flops, multiplexers, carry chains, and more.

At the heart of the FPGA fabric is the *Configurable Logic Block* (CLB), shown in Figure 4. The CLB is the fundamental unit used to implement custom logic. In the Artix-7 architecture, each CLB contains two slices. Each slice integrates several key components, which are highlighted using distinct colors in Figure 4 for clarity: (I) four 6-input Look-Up Tables (LUT6s), shown in blue, which can implement arbitrary six-input Boolean functions;

(II) eight flip-flops (only half are drawn in the diagram), marked in yellow, used to store state or enable sequential logic; (III) the dedicated carry logic (e.g., the `CARRY4` primitive), highlighted in red, designed for efficient arithmetic operations; and (IV) the multiplexers and routing structures, indicated in green, which provide interconnection within the slice and between neighboring CLBs.

This flexible structure allows CLB slices to be configured to implement combinational logic, sequential circuits, arithmetic units, registers, and many other hardware constructs. During implementation, Vivado maps each function in the Verilog design to the appropriate CLB resources based on its logic and routing requirements. If a design exceeds the available resources of the target device (in this work, the Artix-7 XC7A100T-2FTG256), implementation will fail.

Vivado also issues warnings for structures that may cause timing or reliability issues, including deep logic chains or suspected ring oscillators. Once synthesis and implementation complete successfully, Vivado generates a *bitstream*, i.e. the binary configuration file that programs the FPGA, defining the behavior of every logic element and routing connection in the design.

Power Measurement Tools: In addition to Vivado, this project uses the *Nordic Power Profiler Kit (PPK2)* [Nor25] to measure current and regulate the voltage supplied to the FPGA board. The PPK2 is connected in series with the power supply to the CW305 FPGA board, enabling high-resolution current monitoring during active circuit operation. To measure the actual voltage experienced by the Artix-7 FPGA, we also utilize the built-in 7-segment display on the CW305 board. This display reflects real-time voltage levels across the FPGA's core power rail. The nominal operating voltage is 1.0 V. However, during experiments, the voltage drops are expected if the current draw increases significantly. These voltage drops are a result of internal board-level protection circuits designed to prevent damage from excessive current draw conditions, which may be triggered, for example, by aggressive power hammering circuits.

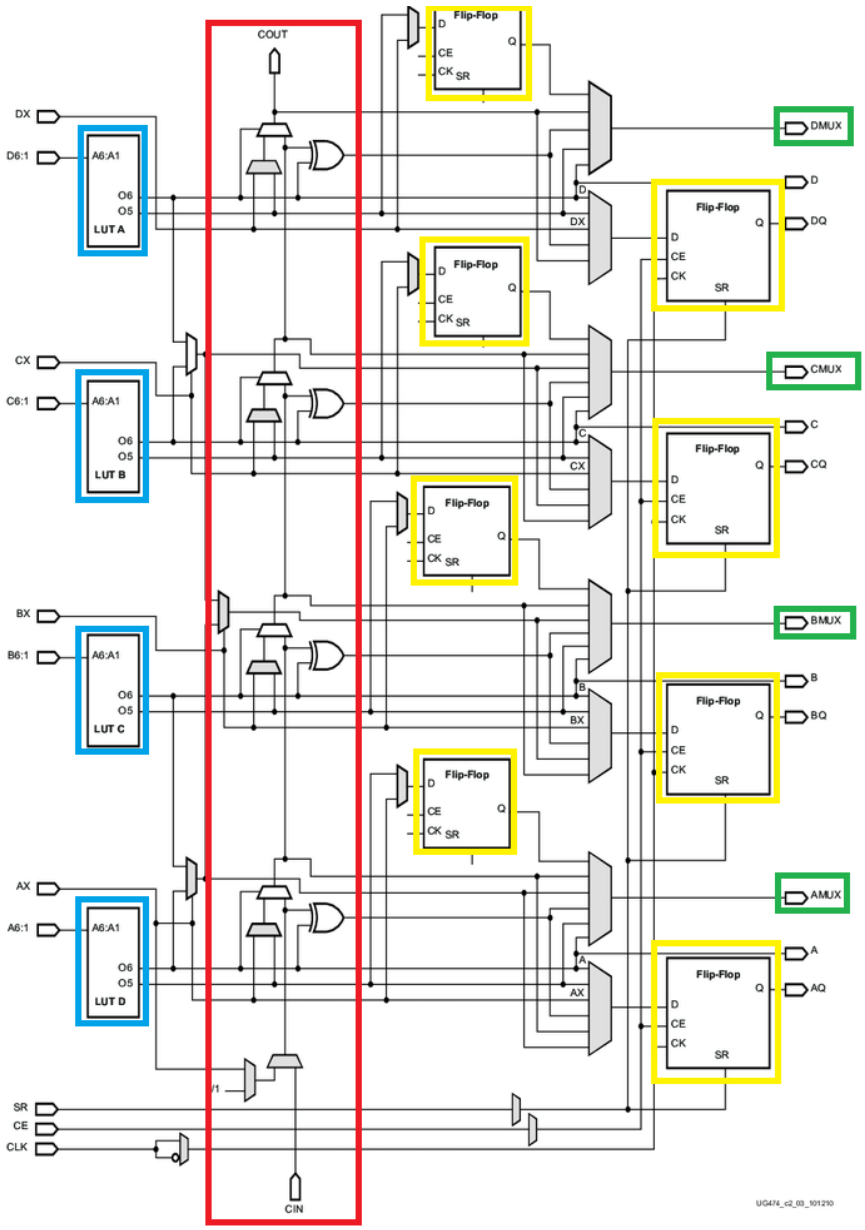


Figure 4: *Diagram of a Xilinx CLB slice (Artix-7 architecture) with the key components marked, in blue the LUTs, in red the Carry chain, in yellow the Flip-Flops and in green the path to the MUXes*

3 Related Work

This section focuses on the taxonomy and risk profile of power hammering circuits, taken from prior work, particularly the *FPGADefender* study [LMG+20]. We revisit the design space outlined in Table 1, which catalogs several variants of ring oscillators, evaluating their power consumption, switching frequency, delays, Vivado detection success (DRC Warning), and resource utilization. Understanding these designs is essential for determining which circuits present the highest risk and which warrant closer examination in our experimental evaluation.

3.1 Power Hammering Circuits

Among the various hardware-level threats facing FPGA designs, power hammering circuits represent a stealthy and destructive class of attacks. These circuits are designed to maximize the switching activity within the FPGA fabric, dramatically the increasing dynamic power consumption while performing no useful computation, resulting in degradation of power integrity, potential system instability, or even physical damage to the device.

- **Inverters**: The first six circuits in Table 1 use a single LUT configured as an inverter (NOT gate) within a ring oscillator loop. While they perform the same logic function, they differ in how the LUT configuration utilizes internal resources. This results in noticeable variations in both power consumption and switching frequency. This highlights how even small changes in logic routing can significantly affect the circuit behavior.

- **Dual Inverters**: These circuits (row 7 in Table 1) use two inverter loops connected to a shared multiplexer to create a more complex feedback structure. Interestingly, this setup is uses less power per LUT than the single inverter loop. For example, while the dual inverter loop consumes 8.04 W using two LUTs, averaging to 4.02 W per LUT, compared with 7.32 W per LUT in the more power consuming single inverter configuration.

- **Enhanced Ring Oscillators**: Rather than maximizing the switching frequency, these designs intentionally configure LUTs inefficiently to increase internal power consumption. Such design prioritizes signal routing complexity and toggling activity over raw speed, and can consume substantial power using relatively simple logic resources.

- **Multiplexers (MUXes)**: These circuits (row 9 and 10 in Table 1) are noteworthy because they were *not* detected by Vivado in the *FPGADefender* study. They use MUXes to route feedback in a loop, forming an unconventional oscillator that consumes less power per loop but may evade detection. Due to their stealthy nature and ease of replication, these circuits represent a significant threat, especially if upscaled to large quantities.

- **Carry Adders**: These circuits (row 11 in Table 1) utilize the `CARRY8` primitive, which is an 8-bit wide arithmetic carry chain used in adders. By feeding the carry output back into the input, an oscillator is formed. Despite being a valid part of arithmetic logic, this configuration can consume high amounts of power and was not detected by Vivado in the FPGADefender study, posing another major concern.

- **Digital Signal Processors (DSPs)**: DSP blocks are large, dedicated hardware components within the FPGA used for efficient arithmetic operations, such as multiply-accumulate (MAC) operations. They are used for signal processing tasks like filtering, convolution, or Fast Fourier Transform. In the context of ring oscillators, feedback paths can be artificially created using DSPs to form high-power circuits. Fortunately, these designs were flagged by Vivado, which suggests some protection is already in place against misuse of these high-power blocks.

- **Latches**: Latches are level-sensitive storage elements that can hold one bit of data. Unlike flip-flops, they continuously respond to input changes when their enable signal is active. In this way, they can introduce instability in circuits when used improperly. Although less common in modern digital designs (where flip-flops are mostly used), latches can still be misused to form parts of power hammering circuits. Notably, these were not detected by Vivado in the FPGADefender study.

- **Flip-Flops (FFs)**: Flip-flops are edge-triggered storage elements and are the standard method of storing the state in synchronous digital systems. In power hammering circuits, FFs can be exploited by feeding a toggling signal into both the `PRESET` and `CLEAR` inputs, forcing constant switching. Fortunately, Vivado correctly detected such circuits in the FPGADefender study, indicating some robustness against FF-based oscillators.

- **Glitch Amplification**: Glitch amplification is a technique where transient, unintended switching events ("glitches") are exploited to increase dynamic power consumption. These glitches often arise from signal race conditions or unbalanced signal paths. In this type of circuits, flip-flops and logic gates are combined in such a way that the number of transitions per clock cycle is artificially increased. This technique can result in elevated power consumption despite not producing continuous signal oscillation like a traditional ring oscillator.

With the behavior and detection characteristics of each aforementioned circuit, particular attention is given to the circuits that previously bypassed detection in Vivado. Especifically, the multiplexer-based, carry logic, and latch-based circuits. These circuits serve as a benchmark for our experiments to evaluate improvements in the Vivado's detection mechanisms.

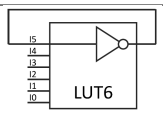
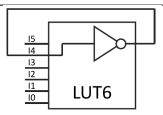
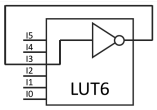
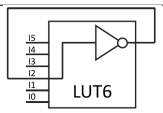
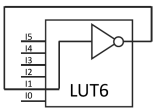
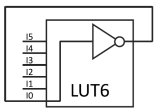
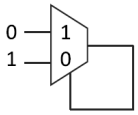
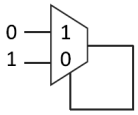
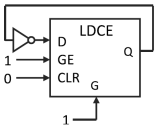
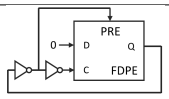
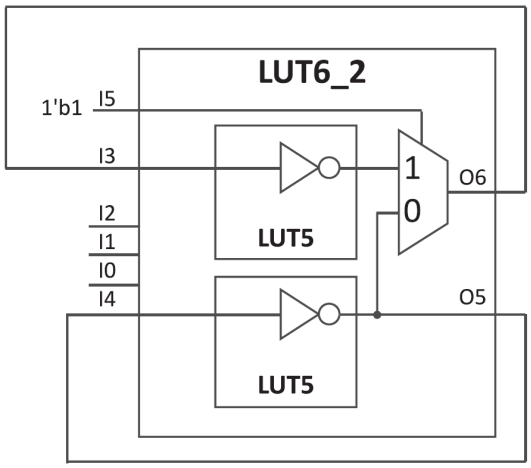
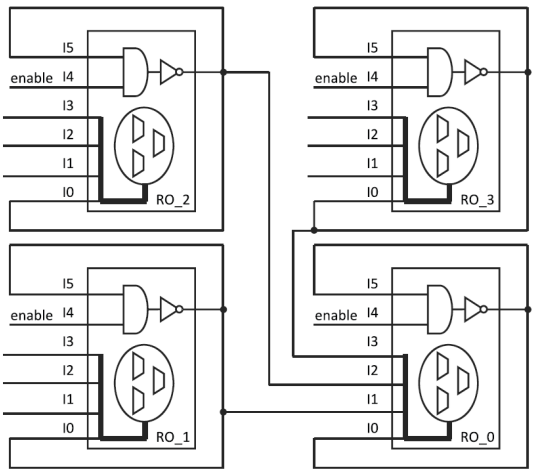
No.	Variants	Schematics	No. of loops	Loop Type	DRC Warning	Report Net Delay	Report Inverter Delay	Expected Frequency	Measured Frequency	Power	WPP
0	Empty design	–	–	–	–	–	–	–	–	2.94W	–
1	RO using LUT6 (I5)		2000	Comb	(LUTLP–1)	49ps	41ps	5556MHz	5882MHz	7.32W (+4.38W)	26.63
2	RO using LUT6 (I4)		2000	Comb	(LUTLP–1)	51ps	66ps	4274MHz	3937MHz	6.84W (+3.90W)	23.69
3	RO using LUT6 (I3)		2000	Comb	(LUTLP–1)	46ps	100ps	3425MHz	3012MHz	5.99W (+3.05W)	18.52
4	RO using LUT6 (I2)		2000	Comb	(LUTLP–1)	50ps	116ps	3012MHz	2488MHz	5.63W (+2.68W)	16.32
5	RO using LUT6 (I1)		2000	Comb	(LUTLP–1)	62ps	150ps	2358MHz	2320MHz	6.59W (+3.65W)	22.21
6	RO using LUT6 (I0)		2000	Comb	(LUTLP–1)	71ps	177ps	2016MHz	1927MHz	6.35W (+3.41W)	20.75
7	Dual-RO from two LUT6 primitive	See Figure 5a	2000 × 2	Comb	(LUTLP–1)	O5: 308ps O6: 54ps	O5: 85ps O6: 100ps	O5: 1272MHz O6: 3247MHz	O5: 1235MHz O6: 2439MHz	8.04W (+5.10W)	31.00
8	Enhanced ROs for power-hammering	See Figure 5b	2000	Comb	(LUTLP–1)	64ps	66ps	3846MHz	1779MHz	9.61W (+6.66W)	40.54
9	RO using MUX7		2000	Comb	X	353ps	112ps	1075MHz	1126MHz	5.01W (+2.07W)	6.30
10	RO using MUX8		2000	Comb	X	211ps	109ps	1563MHz	1681MHz	4.04W (+1.10W)	1.67
11	RO using Carry Logic	See Figure 5c	2000	Comb	X	381ps	104ps	1031MHz	1109MHz	5.14W (+2.19W)	1.67
12	RO using DSP	See Figure 5d	360 × 8	Comb	X	251ps	994ps	402MHz	585MHz	4.53W (+1.59W)	0.27
13	RO using latch		2000	Non-Comb	X	173ps	96ps	1859MHz	1706MHz	5.14W (+2.19W)	13.35
14	RO using flip-flop		2000	Non-Comb	(PDRC–153; PLHOLDVIO–2)	X	X	X	555MHz	5.26W (+2.32W)	7.05
15	Glitch amplification	See Figure 5e	2000	Non-Comb	(PDRC–153; PLHOLDVIO–2)	X	X	X	481MHz	8.05W (+5.10W)	10.35

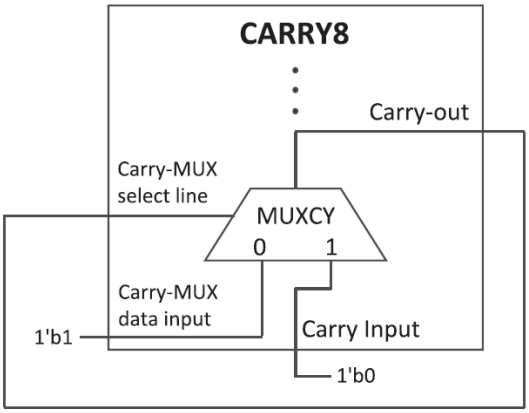
Table 1: *Table 2 from the FPGADefender paper[LMG+20] showing the different types of ring oscillators and their performance as power hammering circuits*



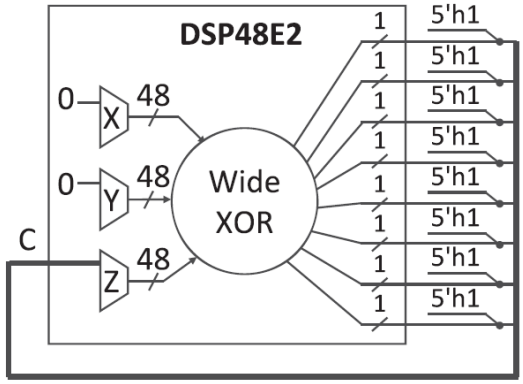
(a) Schematic using dual LUTs for power hammering



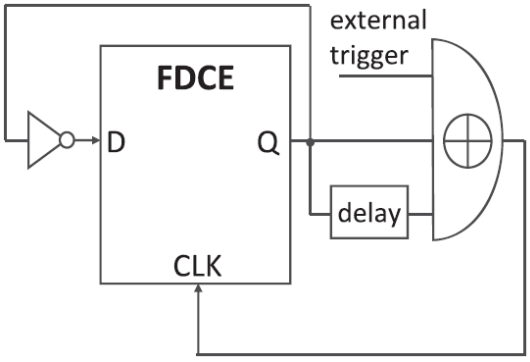
(b) Schematic for enhanced ROs optimized for power-hammering



(c) Schematic for a RO using carry-logic



(d) Schematic for a RO using a DSP



(e) Schematic for a RO using glitch amplification

Figure 5: Extra table for increased readability for schematics

3.2 Security Concerns in FPGA Design

While this thesis primarily investigates power hammering circuits, they exist within a broader landscape of security vulnerabilities affecting FPGA-based systems. FPGAs, by virtue of their reconfigurability and complexity, are exposed to a wide array of attack vectors. Some target the configuration process itself, others exploit physical side channels, and some aim to reverse-engineer or tamper with deployed designs.

Bitstream Attacks: The paper by Michail Moraitis [Mor23] demonstrates how attackers can reverse-engineer and manipulate FPGA bitstreams to extract intellectual property or insert malicious circuits. It also highlights how weak or absent encryption leaves FPGAs vulnerable during configuration. This is closely related to our work, as both bitstream manipulation and power hammering rely on the ability to hide malicious circuits within a valid configuration file. However, while bitstream attacks focus on functional subversion, our work targets resource abuse and physical side-effects. The cited work emphasizes functional integrity but does not explore how malformed bitstreams might affect the FPGAs physical stability. Our research contributes by showing that even semantically valid but malicious configurations can cause instability in FPGAs, expanding the scope of bitstream-based threats.

Side-Channel Attacks: Mark Zhao and G. Edward Suh [ZS18] explore how attackers can extract sensitive information from FPGAs by analyzing power consumption, timing variations, and electromagnetic emissions during operation. The paper provides an overview of physical leakage as a threat to confidentiality. Our research shares a perspective on the physical aspect of the FPGA, but instead of focusing on information leakage, we focus on how similar leakage pathways, like power consumption and switching activity, can be used for denial-of-service attacks via power hammering circuits. However, their work looks only at confidentiality risks and not at situations where an attacker aims to disrupt or damage the device. Our work instead explores how the same physical effects like power consumption and circuit switching activity can be pushed to cause instability. This shifts the focus from using side channels to read information to using them as a way to damage a FPGA.

- **Power Analysis (PA) Attacks**: Schellenberg et al. [SGMT18] details how adversaries can use simple or differential power analysis to recover cryptographic keys by observing power behaviour over time. While our work does not focus on key recovery, it shares perspective on the same observation: power consumption can reveal, or be influenced by, hidden circuit behavior. Both approaches require circuits that produce measurable power fluctuations. This prior work does not consider how purposely designed high-power consumption circuits can disrupt the power behaviour. Our results show that such circuits can both hide sensitive activity and serve as an intentional payload for causing interference.

- **Timing Attacks**: Mukherjee et al. [MSA+21] also illustrates how attackers can infer secrets based on the execution time of operations that depend on internal data. This concept relates to our research in that timing-sensitive designs often have irregular or unintended switching behavior, something power hammering circuits can exploit or imitate. However, the focus remains on inferring secrets, not on stability or damage. Our work shows how abnormal switching patterns can be deliberately induced not to steal secrets, but to destabilize power delivery systems.

- **Electromagnetic (EM) Attacks**: Mulder et al. [DMBO+05] explore how different FPGA logic blocks emit distinct EM signatures, which can be used to recover information using both direct and statistical analysis techniques. This research is somewhat related to ours: both highlight how low-level physical behaviors can be exploited. In fact, EM emissions could serve as an indicator of suspicious high-activity logic like power hammering circuits. Nonetheless, their work treats EM analysis mainly as a passive way to extract information. In contrast, we view these physical emissions as signs of malicious activity, which opens up new possibilities for detecting and preventing hardware misuse.

4 Hardware and Software Setup

To evaluate the power consumption behavior of various potentially malicious FPGA circuits, we developed an experimental setup involving both hardware instrumentation and software tooling. The setup consists of a CW305 FPGA development board featuring an Artix-7 XC7A100T-2FTG256 chip, paired with the PPK2 to measure current draw with high resolution. Software support is provided through Vivado for synthesis and implementation, and the PPK2's native software for high-precision power analysis. Each circuit follows a consistent workflow that guides it from a concept design to measurements, as visualised in Figure 6.
This pipeline-based workflow ensures a structured and repeatable development process, from initial circuit creation in Vivado to final measurements on the physical FPGA. This infrastructure enables us to deploy a variety of power hammering circuits onto the FPGA, observe their effects in real-time, and quantify their impact on the supplied voltage and current. The following sections detail the experimental goals, design methodology, and testing procedures.

Figure 6: *The design and evaluation workflow used for each ring oscillator circuit.*

4.1 Experimental Goals

To design meaningful experiments, it is important to clearly define the objectives and the knowledge we aim to gain. The primary goal of these experiments is to analyze the power consumption behavior of the Artix-7 FPGA when subjected to power hammering circuits. This is accomplished by recreating and implementing several power hammering circuits described in the *FPGADefender* study, and deploying them on our target FPGA. Specifically, we aim to determine: (I) the number of power hammering circuit instances required to cause system instability, performance degradation, or system overload from power hammering circuits; (II) the corresponding power consumption and oscillation frequency of each design; and (III) how effectively the current version of Vivado detects or flags these circuits during synthesis and implementation.

An additional goal is to evaluate how clearly Vivado communicates any warnings or detection results. Previous work [LMG+20] reports that, warnings are sometimes buried among other messages or logs, while in other cases, they are prominently displayed as critical alerts. Understanding the consistency and visibility of such warnings is essential to assess the practical security of the Vivado software.
Together, these goals will help us evaluate both the hardware vulnerability of the Artix-7 FPGA and the progress (if any) made by Vivado in identifying malicious circuit patterns.

4.2 Circuit Design and Simulation

The design of the power hammering circuits begins with the goal of recreating known circuits from the *FPGADefender* paper. While the process seems straightforward, implementation on the Artix-7 FPGA presents several challenges due to architectural differences and unsupported primitives. For instance, the authors of the FPGADefender paper employed the `CARRY8` primitive, which is not available on the Artix-7 series. As a result, our design is adapted using the `CARRY4` primitive, functionally similar but supporting fewer bits. Although this change is unlikely to skew the results significantly, it introduces some debugging overhead.

Fortunately, most other required components are available, including LUTs, multiplexers, flip-flops, and latches. All designs are written in Verilog HDL and synthesized using Vivado. To prevent Vivado from optimizing out the ring oscillator logic, which often lacks meaningful output, we add dummy outputs and wired each oscillator's final output to a probe pin confirming activity. These dummy connections preserve the integrity of the circuits during synthesis and simulation. Before implementing the actual power hammering circuits, a baseline power measurement is taken with the FPGA in an idle state. This provides a reference point for evaluating power increases caused by the power hammering circuits.

Single Inverter Loop: The simplest oscillator consists of a single Look-Up Table (LUT6) configured as a NOT gates (see Figure 7), with its output fed back into one of its inputs. Vivado's LUT6 primitives support six input signals (I0-I5) and one output (O), implementing any Boolean function with up to six inputs [AMD21, Section: LUT6]. Internally, a 64-bit truth table is programmed into the LUT's SRAM-based configuration cells. When the output is looped back into the input, a self-sustaining feedback loop is formed. The Verilog implementations for all six variants are provided in Appendices 7.1.

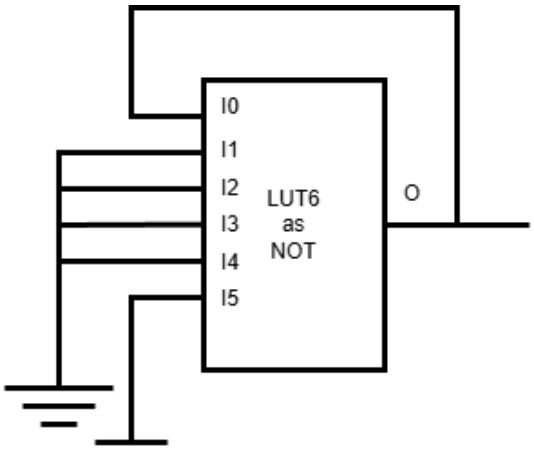


Figure 7: *Inverter-based loop using LUT6*

Six circuit variants are designed and simulated by connecting the feedback path to each of the six inputs individually (I0–I5). Although the logical function is the same in all cases, the routing delay and physical layout differ across inputs, resulting in observable differences in the circuit switching frequency and delays. These variations arise from the fact that different input lines route through slightly different paths within the LUT itself.

Dual Inverter with MUX: This design connects two LUT-based inverter loops to a `MUXF7` primitive (Figure 8). The `MUXF7` acts as a two-input selector, with the select signal (`S`) determining whether input I0 or I1 is passed to the output [AMD21, Section: MUXF7]. Internally, it is implemented using transmission gates and control logic within a Configurable Logic Block (CLB), along with associated routing resources. By statically assigning `S = 1`, the MUX constantly selects the output of one LUT2 table and feed it back to the input of the same table, formaing a loop. This increases the resource utilization and signal toggling activity without significantly altering the logical behavior. Internally, the MUX adds extra switching logic that toggles even when unused paths remain idle at the logic level. The corresponding Verilog description is included in Appendix 7.2.

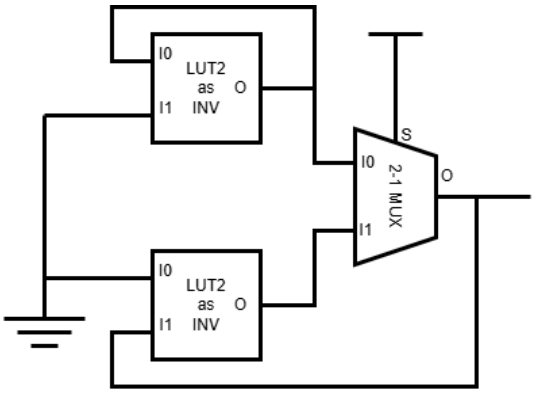


Figure 8: *Dual inverter with MUXF7*

Extended Ring Oscillators: The circuit design shown in Figure 9 implements an intentionally inefficient ring oscillator using six `LUT6` primitives [AMD21, Section: LUT6]. Unlike simpler inverter loops that aim for maximum frequency, this circuit is configured to maximize internal dynamic power draw by exploiting redundant input toggling and internal routing. The first five `LUT6` units are configured as inverter loops with an additional enable input. Each operates as a controlled NAND gate with self-feedback, where toggling is driven by both the feedback signal and a constantly enabled control line. These five LUTs independently oscillate but do not contribute directly to the output logic of the system. The full Verilog implementation is shown in Appendix 7.3.

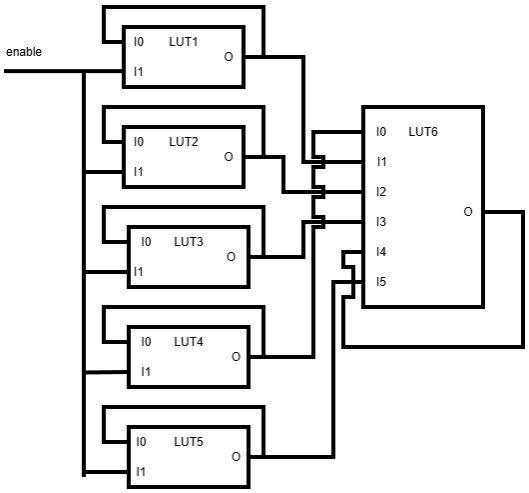


Figure 9: *Extended Ring Oscillator design*

Their outputs feed into the sixth `LUT6`, which is also configured as an inverter loop with enable control. However, the five incoming signals from the five LUTs are wired in such a way that they do not influence the feedback path of the sixth inverter, their input is simply ignored in the configuration the LUT6. Their presence ensures additional signal toggling activity through the use of wider logic input combinations and unnecessary signal switching. The overall purpose of this configuration is not to achieve the highest oscillation frequency possible, but to force all six LUTs into a state of continuous, complex toggling activity. By routing more signals and increasing internal switching capacitance, the circuit elevates the total power consumption well beyond that of the simpler configurations like the inverter which keeps the switching activity only within it's own single LUT at the cost of an large increase in used components.

Multiplexer-Based Oscillators: Oscillators based on `MUXF7` and `MUXF8` primitives are implemented to evaluate multi-level selector logic (see Figure 10) [AMD21, Section: MUXF7, MUXF8]. The `MUXF8` primitive is constructed by cascading two `MUXF7` blocks with an additional select line. These multiplexer stages are built using the FPGA's underlying LUTs (Look-Up Tables), which are reconfigured to act as 2:1 selectors. In a typical implementation, a 6-input LUT can be used to emulate a 2:1 MUX by assigning input-output mappings that respond to a select signal. `MUXF7` and `MUXF8` chains further combine multiple LUTs with fixed internal interconnects inside a configurable logic block (CLB). This structure consumes more LUTs per stage than a simple inverter or logic gate loop, because each multiplexer instance must route multiple data and control paths while still supporting high-speed switching. Each `MUXF8` requires four LUT6 components and two MUXF7s, creating deeply nested signal paths. These configurations can be exploited to create long feedback loops that switch frequently but perform no computation. In previous Vivado versions, these were not detected due to their legitimate multiplexer structure, though they behave as oscillators in this configuration. Their implementations appear in Appendix 7.4.

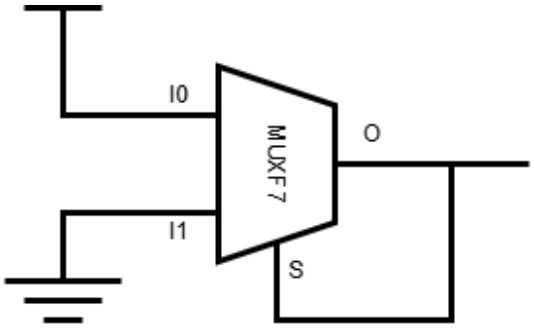


Figure 10: *MUXF7-based oscillator*

Carry Logic Oscillators: The `CARRY4` primitive implements fast carry logic within each Configurable Logic Block (CLB) and is commonly used in arithmetic operations [AMD21, Section: CARRY8]. Internally, it consists of four interconnected MUX-XOR pairs arranged to propagate carry signals between bits in an adder chain. To construct the oscillator, only the first stage of the `CARRY4` primitive is used (as can be seen in Figure 11). Specifically, the carry-in (`CI`) and data-in inputs of the first stage MUX are fixed to a constant '0' and '1', respectively, and the MUX carry-out (`CO`) is routed back to its carry-select (`S`), forming a self-sustaining loop. The remaining three stages in the `CARRY4` block are left unused. This specific configuration simplifies the feedback path while still taking advantage of the primitive's high-speed switching characteristics. Because the carry logic has dedicated routing within the FPGA fabric, the resulting oscillator can toggle with very high frequency. In earlier Vivado versions, this usage was not flagged as problematic, making it a particularly efficient and stealthy option for inducing excessive power draw. The Verilog source code is provided in Appendix 7.5.

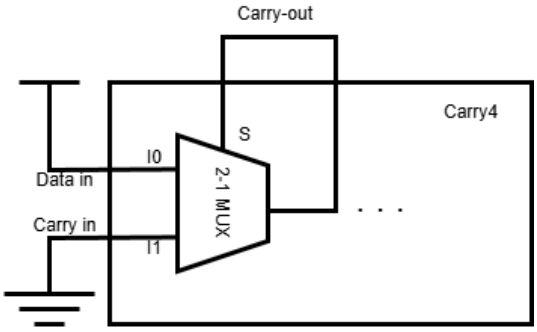


Figure 11: *Carry-chain based oscillator*

Digital Signal Processors (DSPs): DSP slices in Xilinx FPGAs are specialized hardware blocks for arithmetic functions such as multiplication and accumulation. Internally, they include bit multipliers, adder-subtractors, accumulators, and optional pipeline registers. Creating a ring oscillator in DSPs is theoretically possible by feeding output results back into their input stages with combinatorial paths. However, due to their complexity and the lack of a straightforward feedback loop mechanism, no DSP-based oscillators are implemented in this work. Further experimentation and understanding of the DSP block feedback loop options are needed.

Latch-Based Oscillators: Latches such as the `LDPE` are level-sensitive memory elements with asynchronous preset and clear signals [AMD21, Section: LDCE]. Internally, they use transmission gates controlled by the gate (`G`) input signal, which allows data at the `D` input to flow through to the output`Q` when enabled. In our configuration (see Figure 12), we connect `G` and `GE` to logic high and keep the `CLR` input low, allowing the latch to operate continuously. The `Q` output is inverted and fed back to input `D`, forming a feedback loop that toggles as quickly as the routing and gate delays allow. This behavior generates high switching activity across the latch internal path and associated logic slices. The full Verilog code is provided in Appendix 7.7.

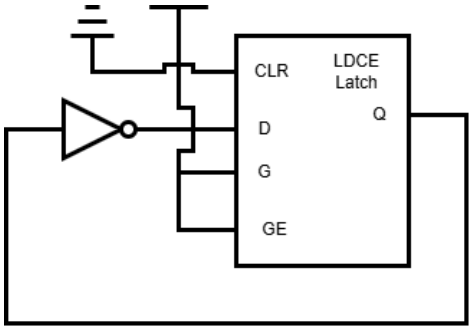


Figure 12: *LDCE latch-based oscillator*

Flip-Flop Oscillators: The `FDPE` flip-flop is an edge-triggered storage element with asynchronous preset and clock enable [AMD21, Section: FDPE]. Internally, it uses a master-slave latch configuration and responds to rising clock edges. Initially, feeding the `Q` output directly into the clock input fails to generate sustained toggling due to timing constraints. However, with further experimentation, a functional oscillator is achieved by constructing a feedback loop using the `PRE` and `CLR` inputs alongside some combinational logic (see Figure 13). The resulting circuit operates without an external clock, instead relying on internal feedback to create a self-sustaining toggling pattern. While the frequency is lower than that of pure LUT-based loops due to clocking and setup/hold delay constraints, the design reliably induces high dynamic power consumption. These oscillators also provide a useful benchmark for analyzing the detection robustness of Vivado. The Verilog listing is included in Appendix 7.6.

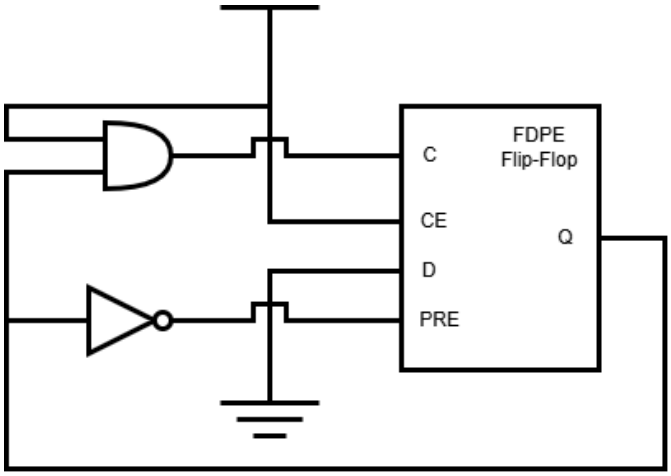


Figure 13: *FDPE flip-flop oscillator*

Glitch Amplification Circuits: These circuits (for example see Figure 14) exploit asynchronous signal transitions and unbalanced logic paths to generate high-frequency switching without forming traditional oscillators. Internally, they rely on races between signal edges that arrive at slightly different times at a logic gate (e.g., XOR or MUX), resulting in glitches, i.e. brief unwanted transitions. While difficult to simulate and control, these glitches can collectively increase dynamic power consumption. They are harder for Vivado to model accurately, as timing violations are usually filtered or averaged out during analysis. As such, they present a non-traditional power threat.

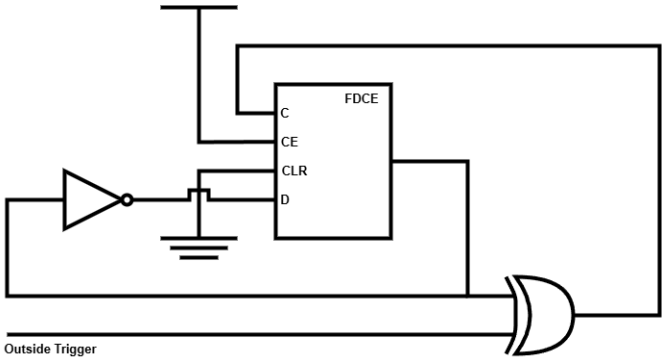


Figure 14: *Glitch-based design*

4.3 Spatial Testing Circuit

This circuit is designed to study the spatial behavior of the FPGA fabric under an increasing number of instantiated power hammering circuits. The FPGA is divided into eight distinct regions, each associated with a dedicated LED indicator, as shown in Figure 15. Every region implements a circuit that generates a periodic flicker signal by a simple counter. Every flicker signal controls the blinking of the associated LED indicator connected to a specific part of the FPGA as shown in Figure 15. Under normal FPGA operating conditions, all eight LEDs blink asynchronously at consistent, predictable rates, serving as a simple but effective indicator of the circuit stability across the chip. Then, a power hammering workload is introduced by gradually increasing the number of simple inverter loops, instantiated across the FPGA fabric by Vivado, so no loops are added on specific locations. Each loop represents a minimal structure configured to toggle continuously, maximizing the dynamic power consumption and internal switching activity. By scaling the number of active loops, we can observe how localized and global power consumption increase propagates across the device, the Verilog code of this spatial testing circuit can be found in the Appendix under 7.9. This regional testing approach provides a clear visual and empirical method to identify the threshold at which the FPGA's internal timing network and power consumption distribution can no longer maintain a reliable FPGA operation. Beyond this threshold, any measurement or signal output from the FPGA device that affect the blinking frequency of the LEDs can no longer be trusted, as circuit instability, missed clock edges, or transient logic errors begin to occur. The resulting dataset when experimenting with the spatial testing circuit captures not only the total power consumption but

LED	Colour	FPGA Region	Pin
LED0	RED1	X0Y0	T2
LED1	GREEN1	X1Y0	T3
LED2	BLUE1	X0Y1	T4
LED3	RED2	X1Y1	C16
LED4	BLUE2	X0Y2	D13
LED5	GREEN2	X1Y2	B14
LED6	WHITE1	X0Y3	B16
LED7	YELLOW1	X1Y3	C13

Figure 15: *Location of LEDs for spatial testing*

also the onset and spread of functional instability across the FPGA fabric as a function of the number of active power hammering loops.

4.4 Experimental Hardware Setup

With the aforementioned circuits designed and simulated, we describe the hardware setup for the circuit deployment and physical testing. As previously stated, the experiments are performed on a CW305 power measurement board, which hosts the Artix-7 XC7A100T-2FTG256 FPGA. To supplement voltage readings, we also use the PPK2, which enables precise measurement of current, a feature not directly supported by the CW305 board. Both the CW305 and the PPK2 are connected to a laptop over a USB port, which acts as the power source and host for programming and data acquisition. The system draws a maximum current of 1 Ampere at a supply voltage of 1 Volt, giving a theoretical upper limit on the power consumption of 1 Watt. In practice, however, the observed power consumption remains below this limit.

Figure 16 shows the complete hardware setup used in this study. The laptop supplying power and controlling the boards is positioned just outside the image to the right. In the foreground, we can see the CW305 board and the PPK2 connected in series. To obtain the most accurate current readings, the standard convenient setup is bypassed: instead of routing the power through the PPK2 first, the laptop's power is connected directly to the CW305. This configuration minimizes measurement interference while still allowing the PPK2 to monitor the current flow with high precision. In the PPK2 software, a maximum voltage limit of 1000 mV is enforced to protect the FPGA during high-load tests. As power hammering circuits are activated, we expect to see a noticeable voltage drop accompanied by a corresponding increase in current, as the loops force the FPGA to consume more power. Figure 17 provides a closer look at the CW305 board. The RGB LEDs on the left-hand side as well as a few extra LEDs (not on the board in the figure) are used as a visual indicator of the reliable operation of the FPGA as explained in Section 4.3, while the 7-segment display on the right shows real-time voltage readings during tests.

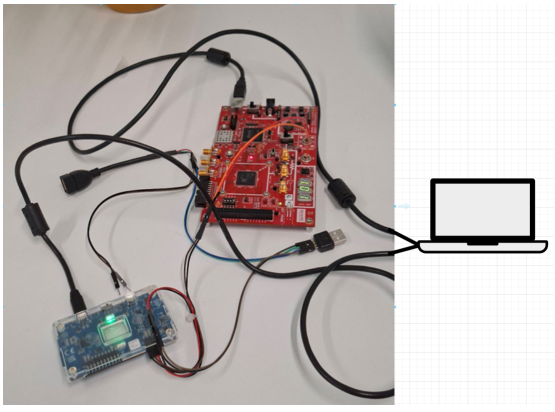


Figure 16: *Full measurement setup including the Nordic PPK2*

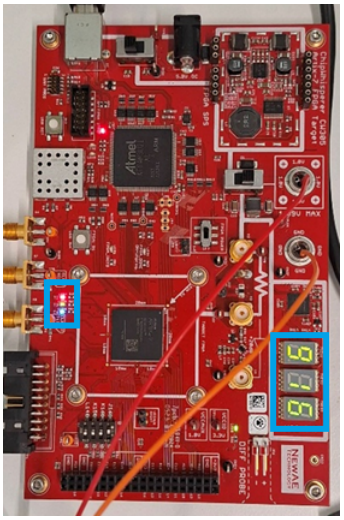


Figure 17: *CW305 with RGB LEDs (left) and 7-segment voltage display (right)*

5 Experimental Results & Observations

This chapter presents the practical evaluation of various power hammering circuits implemented on the Artix-7 FPGA. It begins by detailing the initial challenges encountered within the Vivado design environment, including detection mechanisms and necessary workarounds. Following this, simulation-based frequency estimates are discussed, and real-world power measurements are reported from physical testing on the FPGA. The next chapter reports the measured behavior of the FPGA under increasing switching activity, both through spatial testing and through large-scale single-inverter loop experiments. Finally, the results are analyzed in terms of resource usage, power efficiency, and the conditions that lead to functional instability.

5.1 Challenges in Vivado

The experimental phase began with several unexpected challenges related to Vivado's improved detection of ring oscillators and power hammering circuits. Even with a small number of instances (e.g., 10 loops), Vivado issued warnings during both synthesis and implementation. For larger designs, the software not only generated warnings in the console but also triggered prominent pop-up messages alerting the user for potential risks due to excessive dynamic power consumption. In many cases, these warnings halted the implementation or simulation process entirely, preventing the circuits deployment to the physical FPGA. To continue experimenting, it was necessary to introduce workarounds that would bypass these protective mechanisms. The most effective approach involved modifying the FPGAs constraint file to suppress specific Design Rule Check (DRC) errors. The following line was added to reclassify the "LUTLP-1" warning, used to flag excessive logic toggling, as a non-blocking issue:

```
set_property SEVERITY {Warning} [get_drc_checks LUTLP-1]
```

While this modification allowed synthesis and implementation to proceed, it did not suppress the warnings entirely. Vivado continued to flag the circuits through the Messages tab, console output, and project summary screen, reinforcing the tool's heightened sensitivity to potential power abuse. Beyond synthesis and implementation, Vivado was also used to simulate and evaluate the behavior of each power hammering loop, particularly to estimate their switching frequency. This was done using the *Post-Implementation Timing Simulation*, which incorporates physical delays and routing characteristics of the target FPGA to provide a realistic model of circuit behavior. Since no frequency constraints were applied, each oscillator was allowed to operate at its maximum possible frequency. Due to the lack of high-speed measurement tools such as oscilloscopes or logic analyzers, it was not possible to directly measure the switching the frequency on the hardware. As such, Vivado's timing simulation served as the primary method for estimating frequency. Although these simulations do not fully account for thermal or electrical noise, they offer a reliable approximation and serve as a useful tool for validating whether the loops behave as expected in a real deployment. All power measurements presented in this chapter are based on the average current and voltage recorded over a one-minute runtime for each test configuration. This approach minimizes short-term fluctuations and ensures consistent comparison between different circuit types.

2000 Circuits							1 Circuit				
Circuit	LUTs	MUXes	FFs	Current (mA)	Voltage (mV)	Power (mW)	Simulated Frequency (MHz)	Simulated Delay (ps)	Calculated Frequency (MHz)	Calculated Net Delay (ps)	Calculated Logic Delay (ps)
Baseline	0	0	0	25	975	24	0	0	0	0	0
Inverter A1	2000	0	0	391.7	852	334	398	1256	398	1132	124
Inverter A2	2000	0	0	NA	NA	NA	511	979	519	856	107
Inverter A3	2000	0	0	NA	NA	NA	410	1219	422	1095	90
Inverter A4	2000	0	0	NA	NA	NA	422	1184	439	1060	79
Inverter A5	2000	0	0	NA	NA	NA	1220	410	1445	286	60
Inverter A6	2000	0	0	NA	NA	NA	982	509	1163	385	45
Dual Inverter	4000	2000	0	451.8	836	378	870 (main mux loop 1157 (extra inv loop)	575 432	779 977	380 433	79 60
Extended Inverter	12000	0	0	669.9	759	508	1852 (inv 1) 1134 (inv 2) 1129 (inv 3) 1113 (inv 4) 1754 (inv 5) 1754 (inv 6)	270 441 443 449 285 285	1510 960 956 947 1661 1661	271 442 444 449 241 241	60 79 79 79 60 60
Extended Inverter x333	2000	0	0	384.9	859	331	1852	270	1510	388	45
MUX7	4000	2000	0	319.3	865	276	756	661	791	415	217
MUX8	8000	6000	0	292.7	880	258	613	815	659	542	217
Carry4	2000	0	0	289.1	871	252	1471	340	739	578	99
Flip-Flop	4000	0	2000	404.5	855	346	767	652	218	2052	246
Latch	2000	0	2000	314.3	840	273	688	727	656	510	252

Table 2: *Results FPGA power hammering circuit testing, values for the Inverter A2-A6 are missing due to time constraints and Vivado optimization, a full explanation can be found in section 5.2.*

5.2 FPGA Experimental Results for Power Hammering Loops

Continuing with the actual experiments with the implemented circuits on the FPGA, all results are shown in Table 2. The first column lists each circuit type evaluated in this experiment. For all designs, the listed circuit is instantiated 2000 times to form the complete power-hammering structure implemented on the FPGA. The only exception is the *Extended Inverter ×333* entry, which uses a reduced version of the extended-inverter design containing only 2000 LUTs (333 loops) to allow a fair comparison with the single-inverter circuits. The next three columns report the total FPGA resources used by each design (LUTs, multiplexers, and flip-flops). The `CARRY4` loop appears as a special case, since it relies on the dedicated carry-chain structures already present inside each CLB slice and therefore does not consume additional LUT resources. Following this, three columns list the measured *current*, *voltage*, and resulting *power consumption* for each circuit, obtained using the measurement procedure described in Section 4.4.
The next two columns show the *simulated frequency* and *simulated delay* obtained from Vivado's functional simulation. These values reflect the logical behaviour of the circuits and do not account for the full physical routing characteristics. The relationship between the frequency (MHz) and delay (ps) is as follows:

$$f_{sim} = \frac{1}{2 \cdot \text{Delay}} \cdot 10^6.$$

Finally, the last three columns provide the *calculated* timing parameters derived from the post-implementation timing analysis. Here, the *Net Delay* corresponds to the signal routing delay between components, while the *Logic Delay* refers to the signal propagation delay through the logic element itself. The corresponding calculated frequency (MHz) is determined as:

$$f_{calc} = \frac{1}{2 \cdot (\text{Net Delay} + \text{Logic Delay})} \cdot 10^6.$$

Overall, the simulated frequency and delay columns represent idealized behaviour obtained from Vivado's simulator, whereas the calculated timing values capture the physical routing effects introduced during implementation. It is likely that the simulation does not include all routing effects for many of the circuits, therefore in many cases it reports higher achievable frequencies than the calculated timing analysis.

The measured power results confirm that all tested power hammering circuit types successfully induce substantial dynamic power consumption on the FPGA, with clear variations in both frequency and net delay. Notably, the carry-based oscillator benefits from the dedicated architecture of the CARRY4 primitive. Within each carry block, the feedback path leverages its own internal multiplexer (MUXCY) to form the loop, resulting in a very short and consistent propagation path compared to using the internal MUXes of the CLB. For the flip-flop-based oscillator, the behavior is more complex. The internal structure of the FDPE element introduces two distinct feedback paths, one through the asynchronous preset (PRE) and another through the data (D) input. Timing analysis reveals that these paths correspond to separate propagation loops with delays of approximately 591 ps and 340 ps, respectively. Together, they make up the overall oscillation characteristics observed in simulation.

Because of this distinction, several circuits show a noticeable difference between their simulated and calculated frequencies or delays. The Flip-Flop circuit, for example, reaches 767 MHz in simulation but 218 MHz in the calculated timing analysis, reflecting the added routing constraints of the physical implementation. Similarly, the CARRY4 oscillator exhibits a large difference between the simulated (1471 MHz) and calculated (739 MHz) frequencies, highlighting how the actual routing structure of the carry chain influences its achievable performance. These differences emphasize that the post-implementation timing results more accurately represent the expected hardware behaviour, while simulated values could illustrate the theoretical limits of the design's logical structure.

Unfortunately, it is not possible to obtain measured power data for the Inverter A2 to A6 configurations. During implementation, the Vivado toolchain consistently optimizes or re-maps the inverter loops in an identical way, regardless of the intended input-to-output routing differences. This behavior caused all variants to be physically realized using the same wiring structure, making meaningful measurement comparisons between different designs impossible. Although manual remapping of inputs can be performed for a single inverter instance to achieve distinct loop configurations, allowing for accurate simulation and frequency calculation. Due to the limitations of the automated implementation flow, manual placement constraints for 2,000 instances were outside the scope of this study. As a result, only the simulated and calculated frequency data are included for these designs, while the measured current and voltage values could not be reliably obtained.

We begin by measuring the baseline power consumption of the board without any custom circuit implemented. The idle power consumption is measured at an average current of 25 mA at the default core voltage of 975 mV, resulting in a baseline power of approximately 24 mW.

Single Inverter Ring Oscillators: The single inverter circuit denoted as A1–A6 in Table 2. Their simulated frequencies range from approximately 398 MHz to 1.22 GHz, while calculated values, based on post-routing delays, extend up to 1.445 GHz. The variation between simulated and calculated values indicates how routing and placement significantly affect achievable oscillation frequency. Calculated net delay varies across instances from 286 ps to 1132 ps, while the calculated primitive delay is between 45 ps and 124 ps. The power consumption for 2000 loops was difficult to test as mentioned earlier, but with the A1 inverter having the lowest frequency it is likely to have the lowest power consumption which is at 334 mW with a current of 391.7 mA and voltage of 852 mV. When looking at the data for the frequency, we can deduce that the Inverter A5 has the highest power consumption of the single inverter power loops because it has the highest frequency.

Dual Inverter: The first loop, uses 2000 LUTs and produces a simulated frequency of 1157 MHz and a simulated net delay of 432 ps. The second loop however, incorporates a MUX into the loop, resulting in higher complexity and a LUT usage of an extra 2000 LUTs. The simulated frequency drops to 870 MHz, reflecting the added combinatorial delay. The total delay is simulated at 575 ps and 432 ps, with the calculated primitive delay being 79 ps. This full configuration records a higher power consumption of 378 mW, with a current of 451.8 mA and voltage of 836 mV. The calculated delay and frequency of the two loops are a bit lower compared to the simulation but still remain in an acceptable ratio to the simulated values.

Enhanced ROs: The enhanced ring oscillator group uses 12000 LUTs. Frequency varies widely from 1113 MHz to 1852 MHz, and the simulated net delay is 270 to 449 ps with primitive delay between 60 ps and 79 ps. Power consumption is the highest among all tests at 508 mW, and current draw reaches 669.9 mA, likely due to the high amount combination logic and active switching elements in the design. There is a quite large difference between the calculated and simulated values, and the ratio of calculated versus simulated values does hold up quite well. To evaluate how much the component count contributes to the power increase, the same design was tested with fewer instances, using only 2000 LUTs. This corresponds to 333 extended-inverter loops, each with identical per-loop characteristics as the previous full 12000 size extended inverter. In this reduced configuration, the power consumption drops back to a more typical 331 mW, with a current draw of 384.9 mA.

MUXF7-Based Oscillators: Using 4000 LUTs and 2000 MUXes, because every MUX7 uses 2 LUTs for its input according to the description given by [AMD21, Section: MUXF7, MUXF8]. This circuit achieves a simulated frequency of 756 MHz. with a calculated frequency being 791 MHz. The power consumption is a moderate 276 mW, and the current draw is 319.3 mA.

MUXF8-Based Oscillators: This implementation doubles the logic resource usage of the MUXF7 design because a MUX8 is simply a combination of 2 MUX7s to a new MUX causing 3 MUXes to be used per circuit, totalling 8000 LUTs and 6000 MUXes. The added complexity corresponds to a lower oscillation frequency of 613 MHz, with a calculated frequency of 659 MHz. Despite its larger footprint, it draws a relatively low 258 mW, with 292.7 mA current, implying a trade-off between frequency and overall energy efficiency. The difference between calculated and simulated values are minimal.

Carry Logic Oscillators: This oscillator leverages the FPGAs dedicated carry-chain logic and uses 2000 LUTs and 2000 MUXes, but those 2000 MUXes are all from within each of the Carry4 components itself. It achieves one of the highest simulated frequency in the set of circuits at 1471 MHz, with a simulated delay of 340 ps and a calculated primitive delay of 99 ps. It operates at 252 mW, with a current draw of 289.1 mA, the high frequency and relatively low power consumption suggests that optimized logic paths within the carry chain enable high throughput without excessive power costs. However there does seem to be a large difference between simulated and calculated values.

Flip-Flop Oscillators: Occupying 4000 LUTs and 2000 Flip-Flops, where 2000 LUTs are used for the NAND-gate and 2000 more for the inverter. This design reaches a simulated frequency of 767 MHz and exhibits a simulated delay of 652 ps, with a calculated net delay of 2052 ps. It draws 346 mW at 404.5 mA. The large difference in calculated and simulated delay is likely because of the reason stated earlier with the different paths.

Latch-Based Oscillators: Implemented using 2000 LUTs and 2000 Flip-Flops for the required inverter, this design oscillates at 688 MHz. It has a high simulated net delay of 727 ps. Power consumption is 273 mW at 314.3 mA. The circuit has the slowest logic delay of 252 ps, which causes it to have a low calculated frequency which translates in a relatively low power consumption. The difference between calculated and simulated values are minimal.

5.3 Spatial Test Circuit Evaluation Results

The experimental results obtained from the evaluation of the circuit described in Section 4.3 are summarized in Figure 18. The plotted measurements show the relationship between the current draw (mA) and core voltage (mV) as the number of active single inverter loops increases. At low loop counts, the current rose almost linearly while the voltage dropped gradually, remaining above 900 mV up to approximately 5,000 loops. Beyond this point, both the current and voltage curves diverged sharply, indicating growing stress on the FPGA's internal power delivery network.

The first visible functional disturbance occurred at 14,000 loops, when LED2 (BLUE1, region X0Y1) stopped flickering, suggesting a local timing or voltage fault. At 16,000 loops, LED2 temporarily recovered before failing again at 22,000 loops, reflecting a marginal stability window near the failure threshold. At 26,000 loops, LED5 (GREEN2, region X1Y2) began flickering more slowly than the others, marking the onset of wider timing inconsistency across the fabric. Finally, at 28,000 loops, all LEDs ceased operation, signifying complete loss of reliable power supply.

Beyond this point, the voltage continues to drop slightly, but no meaningful activity is observed, indicating total system instability. These results confirm that as localized loop density increases, power supply degradation first manifests in isolated regions before propagating across the entire device. The early failure of region X0Y1 suggests slightly higher sensitivity to voltage drop or routing delays in that section of the FPGA, while the final collapse illustrates how power hammering circuits can, for example, saturate the entire power grid. The intersection of these effects defines a clear operational boundary: above approximately 28,000 LUT1 loops, the FPGA operation can no longer be considered trustworthy due to its instability and clock distortion that begins at around 14000 loops.

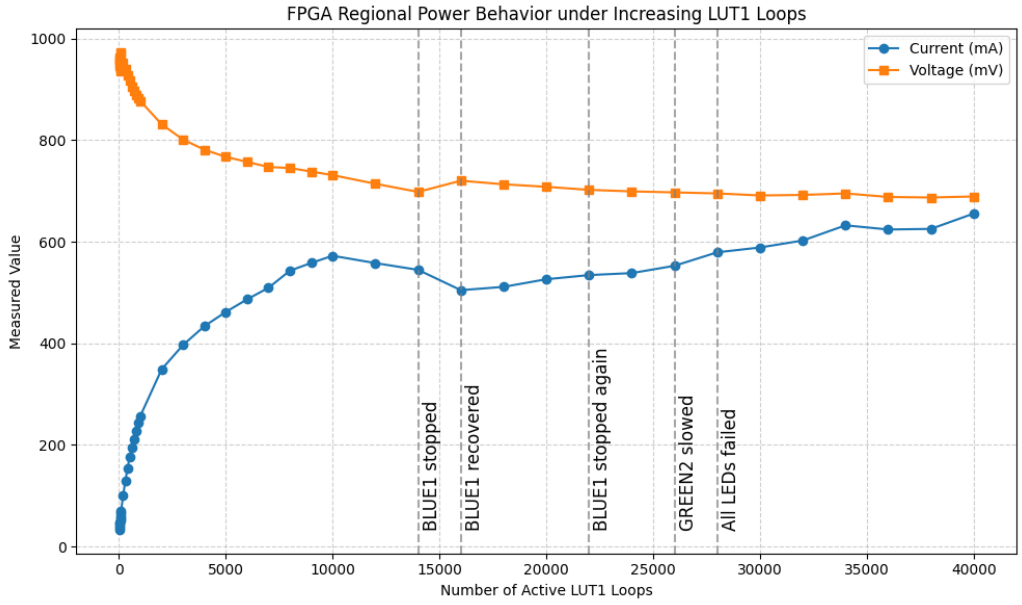


Figure 18: *Measured current (mA, blue) and voltage (mV, orange) versus number of active single inverter loops. Vertical dashed lines indicate key moments of regional failure, including early LED faults (14 000 – 26 000 loops) and full system collapse at 28 000 loops.*

5.4 Single Inverter Loops Power Hammering Effects

To further investigate the contribution of single inverter power hammering circuits to the overall power consumption, a second experiment is conducted focusing exclusively on single inverter loops without any additional counters or LED logic. This experiment shows the power consumption of the FPGA as the number of active single inverter loops increases.

Figures 19 and 20 illustrate the measured current (mA) and core voltage (mV) as a function of the number of active single inverter loops. The first graph shows current and voltage values for up to 500 single inverter loops, while the second graph extends the range to 55,000 single inverter loops.

Using two plots makes the lower loop counts easier to study, because their details are mostly lost in the full-range plot. In the low-loop count range (below 500 single inverter loops), the current draw increases gradually and proportionally with the number of active loops, while the voltage decreases only marginally. A notable spike between 80 and 90 loops is visible in the voltage curve. This sudden jump is likely caused by the power supply's dynamic power scaling, as many modern boards, including the CW305 we evaluate, have a dynamic power scaling which changes the supplied voltage to the FPGA when a certain threshold is crossed.
This interpretation aligns with the observed voltage surge followed by stabilization at higher loop counts. Beyond this region, smaller irregularities appear throughout both graphs, particularly near the mid-range (5,000 - 15,000 loops) and again toward the upper range (around 20000 loops). These fluctuations could be attributed to several potential causes:

- Clock distribution adjustments within the FPGA as internal routing congestion changes.
- Voltage regulation effects, where the onboard power regulators briefly overshoot or undershoot during dynamic load transitions.
- Minor inaccuracies in the current and voltage readings which may also occur when the power profiler changes between different current measurement ranges.

Despite these minor anomalies, the overall trend remains consistent and predictable, i.e. as the number of loops increases, the current rises sharply while the core voltage drops steadily, confirming the relationship between increasing switching activity and increasing power demand.

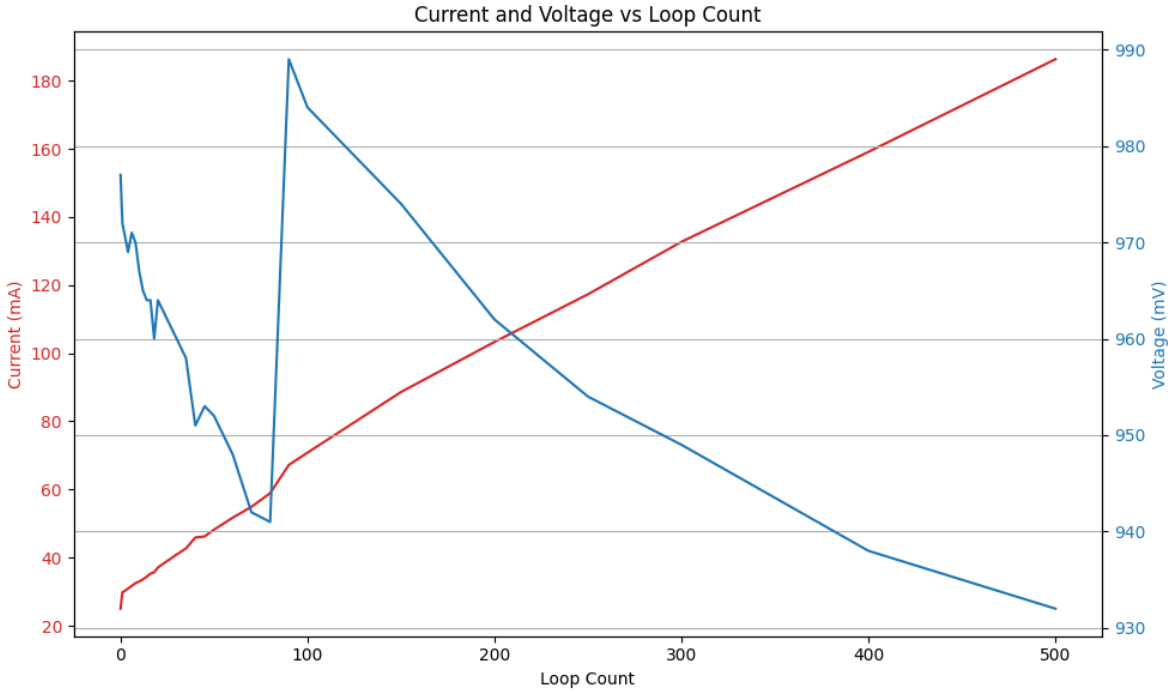


Figure 19: *The first 500 loops and their respective current (mA, red) and voltage (mV, blue) showing a voltage spike around 80-90 loops with an otherwise steady decrease in voltage and a steady increase in current overall*

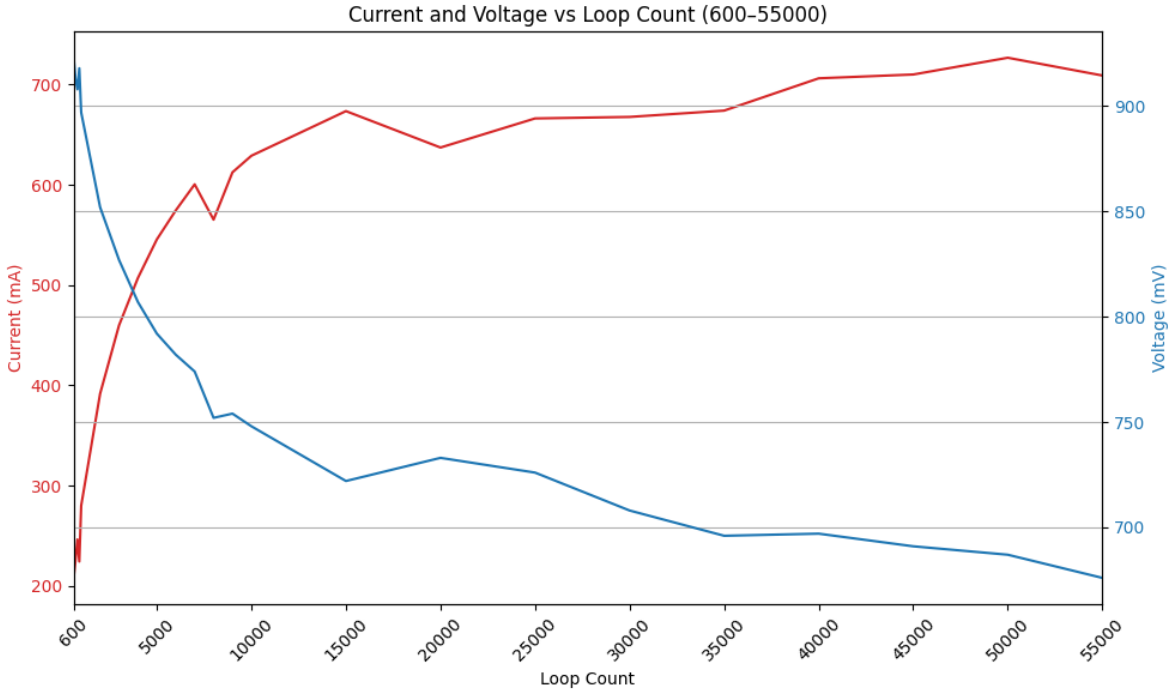


Figure 20: *The full 0 to 55000 loop current (mA, red) and voltage (mV, blue) measurements first showing a sharp increase in current and decrease in voltage before both gradually flatten out as the loops increase*

5.5 Analysis

5.5.1 Ring Oscillator Performance

The results shown in Table 2 provide a detailed comparison between the various ring oscillator implementations in terms of resource utilization (LUTs, MUXes, FFs), switching behavior, and power consumption. Each design demonstrates a distinct balance between frequency, logic delay, and power consumption, revealing the trade-offs between a circuits' architectural complexity and switching frequency.

The single inverter-based oscillators (A1 to A6) form the baseline for the comparison. These circuits rely solely on LUT-based feedback loops and therefore scale predictably with the number of single inverter loops present. Their simulated and calculated frequencies remain in the mid-hundreds of megahertz range, while their power consumption rises steadily with increased loop count. Among these, Inverter A5 and A6 stand out, achieving high simulated frequencies and probable strong power consumption despite using only a single LUT per loop. However, due to the repetitive LUT structure, their per-resource power density is not the optimal, but its simplicity renders it the best option for the single inverter loop testing explained in Section 4.3 and 5.4.

The MUX-based oscillators (MUX7 and MUX8) introduce additional routing and switching elements, but their overall power consumption is lower than the inverter designs (276 mW and 258 mW). This suggests that while these circuits include more logic levels, much of the additional structure does not translate into increased dynamic power consumption, possibly due to partial signal correlation and limited simultaneous signal toggling.

The Carry4-based oscillator proves exceptionally efficient at inducing power stress relative to its size. By exploiting the FPGAs dedicated carry chain, which integrates its own internal multiplexer within each carry block, this design achieves a very short and highly active feedback path. The result is a consistent high-frequency oscillation (1.47 GHz simulated) with minimal routing overhead. Despite using only 2,000 oscillators, it draws 289 mA of current at 871 mV, corresponding to the highest current while using the least amount of resources in the dataset. This potentially makes the Carry4 structure the most effective in terms of the overloading of FPGAs.

The Flip-Flop-based oscillator also demonstrates high total current (405 mA), but its efficiency per element is lower. The Flip-Flop configuration forms two distinct feedback paths, the preset loop (591 ps) and the data (D) loop (340 ps), which divide the switching activity between asynchronous and synchronous transitions. Although this enhances the overall toggling complexity, it also introduces longer path delays and partial clocking dependencies, making the design less power-dense than the carry-based oscillator.

The extended inverter networks (Extended Inverter and Extended Inverter x333) occupy a unique position. By instantiating multiple large inverter chains in parallel, they maximize total logic activity and achieve the highest overall power consumption (508mW). However, this comes at the cost of substantial resource usage (12k LUTs) and reduced per-element power consumption. These designs are ideal for testing global power delivery limits but less optimal in terms of simplicity, because a single loop requires at least 6 LUTs to work, whereas the single inverter loop only uses 1 LUT.

Overall, when comparing the power consumption relative to the resource utilization, the A5-based ring oscillator emerges as the most efficient structure for overloading the FPGA. It achieves high switching activity, minimal routing delay, and a compact footprint and a probable high power consumption, all of which contribute to a disproportionately large impact. The extended inverter (x333) oscillator provides the next highest power consumption due to its chained feedback paths. Importantly, all evaluated circuit variants triggered design rule warnings in Vivado. These included timing and logic warnings, especially related to unrealistic net delays or potential latching hazards. This reinforces prior observations that even structurally valid designs can be flagged due to their behavior at implementation time, likely as part of power-aware design rule checks introduced in recent Vivado versions.

Taken together, these results suggest that the most effective power hammering circuits are not necessarily the ones with the highest complexity, but rather those that maximize switching within minimal area of the FPGA fabric. The Inverter A5 in particular illustrates how efficient, low-delay paths can cause a disproportionally high power consumption.

5.5.2 Single Inverter Loops and Spatial Testing

The results from the power scaling and spatial stability experiments highlight how increasing switching activity directly impacts both the electrical and functional behavior of the FPGA.

In the single inverter loop measurements presented in Section 5.4, a clear trend emerges, as the number of single inverter loops increase, the current increases while the core voltage decreases at a comparable rate. This reflects the inherent limit of the FPGA's power consumption. Since the FPGA cannot exceed its maximum power supply, any increase in switching activity (and thus current draw) is accompanied by a corresponding drop in voltage to prevent overload. Minor irregularities, such as the voltage spike around 80–90 loops, are observable but infrequent and are likely caused by dynamic power management mechanisms or perhaps measurement inaccuracies. Overall, this experiment confirms that the relationship between switching activity and power consumption is largely predictable within the FPGA's operational limits.
The spatial test results in Section 5.3 shows how localized power consumption affects the FPGA's functional stability. Initially, low loop densities produce a similar current increase and voltage decrease, while allowing all LEDs to operate normally. However, once the number of single loop inverters exceed approximately 14,000, early signs of functional instability appear, such as the failing of only LED BLUE1. This indicates a window where the FPGA is still partly stable but nearing the threshold of reliable behavior. As the loop count continues to increase, an additional region of the FPGA begins to fail (e.g., LED GREEN2 at 26,000 loops), following a total functional collapse at 28,000 loops. These observations illustrate that localized power hammering circuits can cause the FPGA to produce results that can not be trusted to be correct.

6 Conclusions and Further Work

In light of the conducted experiments and analyses, this section consolidates the key findings of the research, highlighting the advancements made in detecting power hammering circuits on FPGA platforms. It also discusses the broader security implications these findings have for the FPGA development ecosystem. Finally, the section outlines potential avenues for further investigation aimed at enhancing detection capabilities and addressing the evolving nature of FPGA-based threats.

6.1 Summary of Findings

Firstly, Vivado appears to have improved significantly in its ability to detect power hammering circuits. In contrast to earlier research, where some of the malicious power hammering circuits went undetected, all of our evaluated designs in this study triggered warnings during synthesis or implementation. In many cases, additional steps, such as constraint file modifications, are required to bypass Vivado's safety mechanisms and proceed with running the circuits on the FPGA. This may indicate that the Vivado toolchain has evolved to flag more effectively potentially harmful configurations. Based on the measurements, the power hammering circuits constructed using the `Inverter A5` ring oscillators are considered to be optimal at consuming the highest power with the least FPGA resources. However, these circuits are also clearly detected and heavily flagged by Vivado, making it unlikely they could be deployed without an unsuspecting developer noticing substantial warnings and errors. Across all evaluated circuits, a clear trend does emerge, i.e., higher LUT utilization does not always correspond to proportionally higher power consumption. For example, some complex designs such as the Dual-Inverter and Flip-Flop based circuits utilize significantly more LUTs but only modestly more power is consumed, resulting in lower power-per-LUT efficiency. In contrast, simpler designs like the `Inverter A5` circuits would achieve much higher dynamic power consumption per unit of logic used. This underlines a critical insight, i.e., efficiency in power hammering is driven more by switching density and logic simplicity than raw resource utilization. Designers or attackers seeking to stress the FPGA power supply would benefit more from maximizing signal toggling activity per LUT than from simply scaling up resource usage.

The single inverter loops power-scaling experiments confirm that the current draw increases almost linearly with the number of active loops while the supply voltage steadily drops. This relationship remains until the FPGA device reached its upper operating limit, where the limits of the power supply cause the voltage and current to remain stable as the amount of loops increase.

The spatial (regional) test circuit experiment shows how the electrical stress spreads across the FPGA. As the switching activity increases, circuit timing instability first appears in one region before gradually spreading to the rest of the FPGA device. Full functional failure occurs at around 28,000 active loops and instabilities already around 14,000, indicating the practical limit of the FPGA's reliable operation.

6.2 Implications

As discussed in Section 3 in the context of broader FPGA security concerns, power hammering circuits are not the only threat to FPGA-based systems. Side-channel and bitstream attacks, for example, do not require the execution of malicious logic and can still compromise sensitive information. However, the risks posed by power hammering circuits remain particularly relevant due to their potential for physical damage or system disruption through malicious circuit design alone. Although the evaluated circuits in this study are all successfully flagged by Vivado, there remains a possibility that other, as-yet-unknown designs of power hammering circuits could evade detection. This study focused on compact and previously documented circuits, arguably the most likely to be reused or embedded within larger projects. However, more complex or unconventional loops, including larger circuit topologies or novel combinations of primitives, could still pose a threat. Furthermore, as FPGA architectures evolve, new primitives and features may be introduced that could be misused to create power-abusive loops that cannot be identified by existing detection mechanisms in Vivado.
This highlights the need for ongoing updates to FPGA development tools such as Vivado, not only to detect known potentially dangerous circuit patterns but also to anticipate new forms of abuse as FPGA design techniques advance.
To mitigate future risks, the following improvements could be considered for enhancing power hammering circuit detection in FPGA toolchains:

- **Machine learning-based detection:** Incorporating machine learning models trained on known malicious and benign designs could allow Vivado to identify anomalous structures not covered by hardcoded rules.
- **User-defined power limits:** Allowing developers to set explicit per-region or per-module power constraints could trigger warnings if localized consumption exceeds determined thresholds.
- **Architecture-aware updates:** As new primitives (e.g., enhanced DSP blocks, AI-specific logic) are introduced, detection frameworks should be regularly expanded to evaluate their misuse potential.

Overall, while current detection mechanisms in Vivado are robust for known potentially malicious circuit patterns, proactive and adaptive approaches will be necessary to safeguard future FPGA designs against evolving threats.

6.3 Limitations and Future Work

This research has demonstrated improvements in the detection of power hammering circuits by the Vivado software, but several limitations should be acknowledged. Most notably, all experiments are conducted on a single FPGA device, the Artix-7 (XC7A100T-2FTG256). While representative of many general-purpose applications, FPGAs from other families, such as Kintex, Zynq, or devices from different vendors, feature distinct architectures, resources, and detection mechanisms. As such, the generalizability of our findings to other platforms remains unclear.

Another limitation lies in the scope of circuit types we experimented with. Although the circuits chosen for this study are based on known patterns and are likely candidates for malicious reuse, the space of possible power hammering circuits is much broader. Several attempts were made to explore and develop novel power hammering circuits, but these did not lead to viable new implementations within the limited timeframe of this research.

Future work could expand this investigation by experimenting with a wider variety of FPGA families and architectures as well as exploring more advanced or obscure circuit designs. With increased experience and understanding of low-level hardware behavior, it may be possible to develop entirely new types of power hammering circuits. In addition, integrating some of the proposed improvements, such as anomaly detection, user-defined constraints, or machine learning methods, into Vivado or similar tools could offer further insights into how security mechanisms may be strengthened.

Ultimately, continued research into FPGA-level vulnerabilities remains essential to ensuring the security and reliability of future reconfigurable hardware systems.

References

[AMD21] AMD Xilinx. *UltraScale Architecture Libraries Guide (UG974)*. AMD, 2021. Available at: https://docs.amd.com/r/en-US/ug974-vivado-ultrascale-libraries.

[AMD25] AMD. AMD Vivado Design Suite. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html#overview, 2025. Accessed: 2025-06-06.

[BGG24] Allen Boston, Roman Gauchi, and Pierre-Emmanuel Gaillardon. Secure efpga configuration: A system-level approach. In Iouliia Skliarova, Piedad Brox Jiménez, Mário Véstias, and Pedro C. Diniz, editors, *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pages 151–165, Cham, 2024. Springer Nature Switzerland.

[DMBO+05] E. De Mulder, P. Buysschaert, S.B. Ors, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede. Electromagnetic analysis attack on an fpga implementation of an elliptic curve cryptosystem. In *EUROCON 2005 - The International Conference on "Computer as a Tool"*, volume 2, pages 1879–1882, 2005.

[IEE17] IEEE Spectrum. Chip Hall of Fame: Xilinx XC2064 FPGA. *IEEE Spectrum*, 2017. Accessed: 2025-06-06.

[LMG+20] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. Fpgadefender: Malicious self-oscillator scanning for xilinx ultrascale + fpgas. *ACM Trans. Reconfigurable Technol. Syst.*, 13(3), September 2020.

[Mor23] Michail Moraitis. Fpga bitstream modification: Attacks and countermeasures. *IEEE Access*, 11:127931–127955, 2023.

[Mou25] Mouser Electronics. NAE-CW305-04-7A100-0.10-X. https://mou.sr/43Qsq6B, 2025. Accessed: 2025-06-06.

[MSA+21] Shyamapada Mukherjee, Swapnanil kr Saikia, Stuti Anand, Ritu Chouhan, and Hiresh Das. A counter measure to prevent timing-based side-channel attack on fpga. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pages 983–988, 2021.

[Nor25] Nordic Semiconductor. Power Profiler Kit II. https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2, 2025. Accessed: 2025-06-06.

[SGMT18] Falk Schellenberg, Dennis R.E. Gnad, Amir Moradi, and Mehdi B. Tahoori. An inside job: Remote power analysis attacks on fpgas. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1111–1116, 2018.

[SNM20] Markku-Juhani O. Saarinen, G. Richard Newell, and Ben Marshall. Building a modern trng: An entropy source interface for risc-v. In *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, ASHES'20, page 93–102, New York, NY, USA, 2020. Association for Computing Machinery.

[Ver24] Ieee standard for systemverilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)*, pages 1–1354, 2024.

[ZS18] Mark Zhao and G. Edward Suh. Fpga-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244, 2018.

7 Appendix

7.1 Single-LUT Oscillators

```
module RO_LUT1(
 output wire out,
 input wire prev_out, // Still present to prevent optimization
 input wire enable
);

wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6#(
 .INIT(64'h55555555FFFFFFFF) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(in),
  .I1(1'b0),
  .I2(1'b0),
  .I3(1'b0),
  .I4(1'b0),
  .I5(enable),
  .O(out)
 );
endmodule


module RO_LUT2(
 input wire enable,
 output wire out,
 input wire prev_out // Still present to prevent optimization
);
wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h33333333FFFFFFFF) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(1'b0),
  .I1(in),
  .I2(1'b0),
  .I3(1'b0),
  .I4(1'b0),
  .I5(enable),
  .O(out)
 );
```

```
endmodule


module RO_LUT3 (
 input wire enable,
 output wire out,
 input wire prev_out // Still present to prevent optimization
);
wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h0F0F0F0FFFFFFFFF) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(1'b0),
 .I1(1'b0),
  .I2(in),
  .I3(1'b0),
  .I4(1'b0),
  .I5(enable),
  .O(out)
 );
endmodule


module RO_LUT4 (
 input wire enable,
 output wire out,
 input wire prev_out // Still present to prevent optimization
);
wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h00FF00FFFFFFFFFF) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(1'b0),
  .I1(1'b0),
  .I2(1'b0),
  .I3(in),
  .I4(1'b0),
  .I5(enable),
  .O(out)
 );
endmodule
```

```
module RO_LUT5(
 input wire enable,
 output wire out,
 input wire prev_out // Still present to prevent optimization
);
wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h0000FFFFFFFFFFFF) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(1'b0),
  .I1(1'b0),
  .I2(1'b0),
  .I3(1'b0),
  .I4(in),
  .I5(enable),
  .O(out)
 );
endmodule


module RO_LUT6 (
 input wire enable,
 output wire out,
 input wire prev_out // Dummy to prevent optimization
);
wire in;
assign in = out;
 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h55555555FFFFFFFF)
 ) lut_inst (
  .I0(enable), // A1
  .I1(1'b0), // A2
  .I2(1'b0), // A3
  .I3(1'b0), // A4
  .I4(1'b0), // A5
  .I5(in), // A6
  .O(out)
 );
endmodule
```

7.2 Dual Inverter Oscillator

```
module DUAL_INV(
```

```
 input wire dummy_in,
 output wire out,
 input wire enable
);
// Oscillator 1 (self-looping)
wire osc1_lut_out;
wire osc1_lut_in;

// Oscillator 2 (driven by mux output)
wire osc2_lut_out;
wire osc2_lut_in;

 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut1 (
  .I0(osc1_lut_in),
  .I1(enable),
  .O(osc1_lut_out)
 );

assign osc1_lut_in = osc1_lut_out;
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut2 (
  .I0(osc2_lut_in),
  .I1(enable),
  .O(osc2_lut_out)
 );

// MUXF7 selects between two oscillators
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 MUXF7 mux_inst (
  .I0(osc1_lut_out),
  .I1(osc2_lut_out),
  .S(1'b1), // always selecting I1 = osc2
  .O(out)
 );
assign osc2_lut_in = out;
endmodule
```

7.3 Enhanced Ring Oscillator

```
module RO_LUT6_Chain (
```

```
 input wire enable,
 output wire out,
 input wire prev_out
);
// Internal oscillator signals
wire ro1_out, ro2_out, ro3_out, ro4_out, ro6_out;
wire lut1_inputs;
assign lut1_inputs = lut1_output;

 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut1 (
  .I0(lut1_inputs),
  .I1(enable),
  .O(lut1_output)
 );

wire lut2_inputs;
assign lut2_inputs = lut2_output;
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut2 (
  .I0(lut2_inputs),
  .I1(enable),
  .O(lut2_output)
 );

wire lut3_inputs;
assign lut3_inputs = lut3_output;
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut3 (
  .I0(lut3_inputs),
  .I1(enable),
  .O(lut3_output)
 );

wire lut4_inputs;
assign lut4_inputs = lut4_output;
```

```
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut4 (
  .I0(lut4_inputs),
  .I1(enable),
  .O(lut4_output)
 );

wire lut5_inputs;
assign lut5_inputs = lut5_output;
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // out = ~in when enable = 1, else 1
 ) lut5 (
  .I0(lut5_inputs),
  .I1(enable),
  .O(lut5_output)
 );

wire lut6_inputs;
assign lut6_input = lut6_output;
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 LUT6 #(
 .INIT(64'h0000FFFFFFFFFFFF)
 ) lut6 (
  .I0(lut4_output),
  .I1(lut1_output),
  .I2(lut2_output),
  .I3(lut3_output),
  .I4(lut6_input),
  .I5(lut5_output),
  .O(lut6_output)
 );
assign out = lut6_output;
endmodule
```

7.4 Multiplexer-Based Oscillators

```
module MUXF7_RO (
 input wire dummy_in,
 output wire ro_out
);
```

```
reg feedback_delayed;
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 MUXF7 mux_inst (
  .I0(1'b1),
  .I1(1'b0), // Always 0
  .S(feedback_delayed), // Always select feedback
  .O(ro_out)
 );
 always @ (ro_out) begin
  feedback_delayed = ro_out;
 end
assign feedback = ro_out;
endmodule


module MUXF8_RO (
 input wire dummy_in,
 output wire ro_out
);
reg feedback_delayed;

 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 MUXF8 mux_inst (
  .I0(1'b1),
  .I1(1'b0), // Always 0
  .S(feedback_delayed), // Always select feedback
  .O(ro_out)
 );

 always @ (ro_out) begin
  feedback_delayed = ro_out;
 end
assign feedback = ro_out; // Feedback to itself
endmodule
```

7.5 Carry-Chain Oscillator

```
module CARRY4_RO (
 input wire dummy_in, // Unused, just to preserve structure
 output wire ro_out,
 input wire enable
);
wire co0;
wire input_co0 = enable & co0;
```

```
// CARRY4 primitive: use first mux as an oscillator
 (* KEEP = "TRUE", DONT_TOUCH = "TRUE" *)
 CARRY4 carry4_inst (
  .CI(1'b0), // Carry input is 0
  .CYINIT(1'b0), // Initial carry is 0
  .DI(4'b0001), // DI[0] = 1, rest unused
  .S({3'b000, input_co0}), // S[0] = CO[0]
  .CO(co0), // CO[0] = output
  .O() // O output not used
 );

assign ro_out = co0;
endmodule
```

7.6 Flip-Flop Oscillator

```
(* KEEP, DONT_TOUCH *)
module FDPE_RO (
 input wire dummy_in,
 input wire enable,
 output wire osc_out
);
wire q_int;
wire out_nand;
wire out_inv;
wire ff_out;

 (* KEEP, DONT_TOUCH *)
 LUT6 #(
 .INIT(64'h55555555FFFFFFFF) // NAND-like behavior gated by enable
 ) lut_nand (
  .I0(enable), // Enable control
  .I1(1'b0),
  .I2(1'b0),
  .I3(1'b0),
  .I4(1'b0),
  .I5(q_int), // Feedback from loop
  .O(out_nand)
 );

 (* KEEP, DONT_TOUCH *)
 LUT2 #(
 .INIT(4'h7) // NOT(out_nand), enable ignored
 ) lut_inv (
```

```
  .I0(enable),
  .I1(out_nand),
  .O(out_inv)
 );

// FDPE flip-flop for edge-triggered storage
 (* KEEP, DONT_TOUCH *)
 FDPE #(
 .INIT(1'b0)
 ) ff_inst (
  .Q(q_int), // Feedback output
  .C(out_inv), // System clock
  .CE(1'b1), // Clock enable
  .PRE(out_nand), // No async preset
  .D(1'b0) // Data from inverter
 );
// Final output
assign osc_out = q_int;
endmodule
```

7.7 Latch-Based Oscillator

```
(* KEEP, DONT_TOUCH *)
module RO_LATCH (
 input wire init,
 input wire in,
 output wire out,
 input wire enable
);
wire g;
assign g = ~out; // Inverter for feedback loop

 LDCE #(
 .INIT(1'b0)
 ) latch (
  .Q(out),
  .D(g), // Inverted feedback
  .G(1'b1), // Constant 1
  .GE(enable), // Always enabled
  .CLR(1'b0) // No async clear
 );
endmodule
```

7.8 Generator for multiple instances

```
module RO_LUT1Gen (
 output wire ro_out,
 input wire enable
);
 wire [0:0] ro_signals;

 (* KEEP, DONT_TOUCH *)
 RO_LUT1 ro_first (
  .prev_out(1'b0),
  .out(ro_signals[0]),
  .enable(enable)
 );

 genvar i;
 generate
 for (i = 1; i < 1; i = i + 1) begin : ro_gen
 (* KEEP, DONT_TOUCH *)
 RO_LUT1 ro_stage (
  .prev_out(ro_signals[i-1]),
  .out(ro_signals[i]),
  .enable(enable)
 );
 end
 endgenerate
assign ro_out = ro_signals[0];
endmodule
```

7.9 LED blinking code

```
module RO_STAGE(
 output wire out,
 input wire prev_out // Still present to prevent optimization
);
wire in;
assign in = out;

 (* KEEP, DONT_TOUCH *)
 LUT1#(
 .INIT(2'h1) // out = ~in when enable = 1, else 1
 ) lut_inst (
  .I0(in),
  .O(out)
 );
```

```
endmodule

module heartbeat_counter(
 input wire clk,
 output wire led
);

(* KEEP = "true" *) reg ff = 0;
 reg [25:0] counter = 0;
  always @(posedge clk) begin
  counter <= counter + 1;
  ff <= counter[25];
 end
assign led = ff;
endmodule
```