TECHNISCHE UNIVERSITÄT WIEN

Institut für Computertechnik
Institute of Computer Technology

Master's Thesis

submitted by

Stefan Riesenberger

Registration Number 01609405

FPGA Power Estimation

In partial fulfillment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Vienna, Austria, March 13, 2025

Study code:	066 504
Field of study:	Embedded Systems
Supervisor:	Axel Jantsch
Supervisor:	Christian Krieg

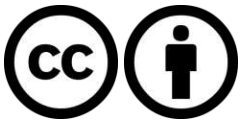
If you find this work useful, please cite it using the following BibTeX entry:

```
@Thesis{Riesenberger2025,
 type        = {Master's Thesis},
 author      = {Stefan Riesenberger},
 title       = {FPGA Power Estimation},
 school      = {Vienna University of Technology (TU Wien)},
 address     = {Gusshausstrasse 27--29 / 384, 1040 Wien},
 year        = {2025},
 month       = {February},
}
```

Contact us:

stefan.riesenberger@alumni.tuwien.ac.at

christian.krieg@alumni.tuwien.ac.at



Acknowledgements

I like to thank everyone that has supported me in the pursuit of my thesis and my studies. Big thanks especially to my university collegues that studied together with me and made the time a lot more enjoyable. My family and partner for nudging me to finish what I have started. Also a big thanks to my supervisor for pushing through with me on this journey.

I also don't want to neglect the open source community which provides great documentation and tools respecting your freedom and not locking you into the eco system of vendors. All this knowledge and tooling is avaliable for anyone to use, which makes the level of entry much lower by removing virtual barriers and restrictions. We are all building on the work of the ones that had come before us, so it is important to make knowledge as accessible as possible.

Abstract

While in the past decade there has been significant progress in open-source circuit synthesis and verification tools and flows, one piece is still missing in the open-source design automation ecosystem: a tool to estimate the power consumption of a design on specific target technologies.

We present a method to characterize target technologies using generic benchmark designs with hardware measurements, whose results are used to fit power models on these target technologies. A setup to gather the power measurement data of the Lattice iCEBreaker FPGA board is provided. For stimulation of designs we resorted to both LFSRs and directly mapping the 12 MHz clock input as the stimulus. We also implement these testbenches on the simulation side to extract valuable information on internal transitions. Benchmark designs to characterize the FPGA are created with low complexity in mind to allow for simple reasoning about their internal behavior and to reduce the need for complex testbenches. The benchmarks include LFSR, ring oscillator, divider and arbiter circuits. Modelling of the FPGA is done as a linear system based on a CMOS model for the internal components. The model fitting is done by solving the linear system fed by benchmarks with classic optimization algorithms. For the assertion of estimation quality we compare the output of our estimator to the vendor tool in three practical use cases, such as a CPU, a stream cipher and a hash algorithm. Hardware measurements of the vendor synthesis have also been taken to provide a ground truth. Applied to the use case benchmarks our estimation works really well. Especially compared to the vendor provided tooling we reach similar accuracy.

Kurzfassung

In den letzten Jahrzehnten gab es signifikanten Fortschritt im Bereich der Open-Source Hardware-Synthese und Verifikation. Ein Teil der jedoch immer noch im Open-Source Design-Automation-Ökosystem fehlt ist ein Tool zum Abschätzen der Leistung welche ein Design auf spezifischer Zieltechnologie benötigt.

Wir stellen eine Methode zur Charakterisierung von Zieltechnologien unter Verwendung generischer Benchmark-Designs mit Hardware-Messungen vor, deren Ergebnisse zur Anpassung von Leistungsmodellen für diese Zieltechnologien verwendet werden kann. Es wird ein Setup zur Erfassung der Leistungsmessdaten des Lattice iCEBreaker FPGA-Boards bereitgestellt. Zur Stimulierung der Designs haben wir sowohl auf LFSRs als auch auf die direkte Abbildung des 12 MHz-Takteingangs als Quelle zurückgegriffen. Wir implementieren diese Testroutinen auch in Simulation, um wertvolle Informationen über interne Transitionen zu gewinnen. Die Benchmark-Designs zur Charakterisierung des FPGAs wurden mit Fokus auf geringe Komplexität erstellt, um einfache Rückschlüsse auf das interne Verhalten zu ermöglichen und den Bedarf an komplexen Testroutinen zu reduzieren. Zu den verwendeten Benchmarks gehören LFSR-, Ringoszillator-, Frequenzteiler- und Arbiter-Schaltungen. Die Modellierung des FPGAs erfolgt als lineares System basierend auf einem CMOS-Modell für die internen Komponenten. Das Modellfitting erfolgt durch Lösen des linearen Systems, das mittels der Benchmarks gefüttert wird, durch klassische Optimierungsalgorithmen. Um die Qualität der Schätzung zu überprüfen, vergleichen wir die Ergebnisse unseres Schätzers mit dem Tool des Herstellers in drei praxisnahen Anwendungsfällen, wie z.B. einer CPU, eines Stream-Cipher und einem Hash-Algorithmus. Es wurden auch Hardware-Messungen der Herstellersynthese durchgeführt, um ihren Grundverbrauch zu erhalten. Angewendet auf die praxisnahen Benchmarks funktioniert unsere Schätzung sehr gut. Insbesondere im Vergleich zu den vom Hersteller bereitgestellten Programmen erreichen wir eine ähnliche Genauigkeit.

Contents

List of Tables

List of Figures

Acronyms

ADC Analog-Digital Converter. 28

CMOS Complementary Metal-Oxide Semiconductor. 27, 55

DUT Device Under Test. 19, 31, 32

FPGA Field Programmable Gate Array. 19, 20, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 42, 43, 52, 53, 55

IC Integrated Circuit. 28

iverilog Icarus Verilog. 21, 40, 41

LFSR Linear-Feedback Shift Register. 11, 13, 27, 31, 34, 39, 40, 41

LUT Lookup-Table. 13, 33, 34, 35, 36, 37, 42, 52, 53

MUX multiplexer. 38

NNLS Non-Negative Least Squares. 13, 19, 50, 51

SVD Singular Value Decomposition. 50

UART Universal Asynchronous Receiver/Transmitter. 52

XPE Xilinx Power Estimator. 17, 18

Chapter 1

Introduction

PRE-PUBLICATION NOTE: We submitted a work-in-progress paper to a workshop to discuss the current state of this thesis with the relevant peer group. The content of Sections 3.2 to 3.4 and chapter 2 are mostly taken verbatim from this paper [6] Stefan Riesenberger and Christian Krieg. "Towards Power Characterization of FPGA Architectures To Enable Open-Source Power Estimation Using Micro-Benchmarks". In: Proceedings of the 3rd Workshop on Open-Source Design Automation (OSDA), 2023, co-hosted with Design, Automation, and Test in Europe (DATE) conference in Antwerp, Belgium, April 17, 2023. This work was entirely done by Stefan Riesenberger under the supervision of Christian Krieg.

1.1 Problem Statement

Modern FPGA hardware design practice quite often requires awareness about the impact of specific design decisions on the power consumption. It is important for environments that are constrained in energy or thermal dissipation to achieve an optimal solution for given requirements. Besides these aspects it is also interesting from an educational point of view to analyze designs and to evaluate power hotspots. This allows for a better understanding and can help the designer to create more optimal designs by default.

Most FPGA vendors provide tools for power estimation in their proprietary toolchains. Such tools vary widely in their capabilities and some are lacking detailed output that would enable deeper understanding of the power consuming mechanisms in a provided design. In addition, these tools are locked behind convoluted licenses and are not easily integrated into open workflows using i.e. Yosys[1].

1.2 Motivating Example

In the hardware design process, the designer is interested in expected power consumption of their design. There exist a multitude of tools from every FPGA vendor for this need of power estimation. These tools differ broadly between supported features and are very often integrated into their EDA suite. The prime example is Vivado with the Xilinx Power Estimator (XPE) tool for power estimation. It provides state of the art features for detailed power estimation of

[1] https://github.com/YosysHQ/yosys

a given design for a Xilinx FPGA platform. Results are provided fine grained per signal, logic type and I/O.

The open toolchain around Yosys provides most features to allow hardware design for Lattice FPGAs. One major lacking point is not supporting power estimation features on its own. It is compatible to be mixed with the tools provided by Lattice, but these tools do not suffice in level of detail about power consumption similar to what XPE offers, as shown in Table 2.1. Proper integration into the Yosys workflow would also be a highly desired aspect.

1.3 Research Questions

1. **Which variation of power consumption can be observed by variations on the design structure?** What main parameters does a designs power consumption depend on and how does the design structure influence it.

2. **What degrees of precision and accuracy can be achieved by design variation in terms of the estimated compared to measured power consumption?** Evaluation of our method's quality by utilizing use case benchmarks that were not used for fitting.

3. **What features and level of detail do hardware vendor power estimation tools from Altera, Xilinx and Lattice provide?** This shall give an overview of existing tools and what new tooling should support to compete with them.

4. **How can the power of a Lattice iCEBreaker FPGA board accurately be measured over a collection of configurations and inputs?** Working out a measurement setup to conduct hardware power measurements on the Lattice iCEBreaker FPGA board, with minor modifications to the board and precise enough measurement accuracy to gather data.

5. **Can low complexity benchmarks be used to analyze power characteristics of a Lattice iCEBreaker FPGA board?** Providing example benchmarks, which show frequency dependent power utilization to enable the characterization of the FPGA's power behavior.

6. **How can activation rates for components be extracted from existing simulation VCD files?** Here we show the extraction of activation rates from VCD files generated by simulation on the post synthesis benchmarks.

7. **How can the parameters of an activation rate based power estimation model be extracted from power measurements on hardware?** Analyzing hardware power measurements to infer the model characteristics. Extracting FPGA component model parameters from these measurements by means of solving optimization.

8. **How well does the provided power estimation solution perform compared to hardware measurements and vendor tools?** A qualitative comparison using hardware measurements between the estimates of the vendor tool and our solution is provided.

1.4 Scientific Contributions

In this chapter we discuss our work based on our research question and provide summary answers to them.

1.4.1 Hardware vendor tool feature levels

The question is about what features and level of detail do hardware vendor power estimation tools from Altera, Xilinx and Lattice provide. We showed in our feature matrix Table 2.1 that some vendors provide extensive tooling for their products and others segregated the capabilities between different tools and product categories.

We also noted that research is quite often using vendor tools as their ground truth for benchmarking, which is not great when other papers already claim that vendor tools are not very accurate in their estimations.

1.4.2 Microbenchmarks on Lattice iCEBreaker

At first we develop a tool to generate simple designs containing only i.e. **DFF**s utilizing tool chain specific features to place elements at the desired location on the Field Programmable Gate Array (FPGA). After this we start constructing regularly structured designs via **generate** statements from simple designs like ring oscillators, arbiter and frequency divider to instantiate arbitrary scaled versions of them.

1.4.3 Power measurements

We analyze the schematic and layout of the iCEBreaker FPGA board and identify positions for measurement shunt resistors to enable power measurements. Then we design a measurement board utilizing current sense amplifiers to amplify the shunt resistor voltage to an easier measurable level. This is then captured by an oscilloscope due to the initial test with a USB measuring card not being sufficient.

1.4.4 Activation rate extraction

Given a specific HDL design in combination with a testbench the activation rate extraction. We first synthesize the design down to the hardware specific gate level and then simulate it with the testbench to generate a waveform of the Device Under Test (DUT). This waveform contains all gates' IO states which in turn is used for activation rate extraction. We achieved this with a tool to parse the waveform file to accumulate the activation rates per gate IO and return the data as a common digestible matrix notation.

1.4.5 Power model, fitting and estimation

For our power model we resort to a linear model based on CMOS gates. A CMOS's total power utilization is described by a static leakage term P_s and a dynamic loss term P_d. P_s represents the power lost from leakage currents times the supply voltage UI_leak. P_d on the other hand describes the capacitive switching losses at a given switching frequency in the from $fC_{cmos}U^2$. Based on this the power of every component made from CMOS gates is just a sum of the power of their CMOS parts.

Based on the mentioned model a linear system in matrix notation $\boldsymbol{P}_{\mathrm{t,B}} = \boldsymbol{Ax}$ is constructed, which is feed by the design parameters power, activation rate and cell usage extracted previously. The linear system is solved by common algorithms like the least squares algorithm given the matrix A is positive-definite. For our use case the classic least squares has the issue of returning negative parameter estimates and it does not function properly on singular input matrices. These problems are solved by using the NNLS algorithm, which is a constraint optimization algorithm. It

solves the given system with inequality constraints to prevent our fitted solution $\boldsymbol{x}$, containing the leakage current and capacitance values, to be negative.

The tools created for fitting the model parameters are with minor tweaking also used to run estimation on a given design. For estimation of an arbitrary design only the calculation of the matrix $\boldsymbol{A}$ is required, due to $\boldsymbol{x}$ being known at this point already, then the power is calculated from the linear system $\boldsymbol{P}_{\mathrm{t,B}} = \boldsymbol{A}\boldsymbol{x}$.

1.4.6 Comparison of our results

Based on the designs and measurements from Section 4.3 we can draw a few comparisons between our implementation and the vendor tooling. Unfortunately the biggest and most complicated benchmark based on the Picorv32 is not possible to be placed and routed onto our FPGA with the vendor tool, due to high resource utilization. This is due to the design using some architecture specific cell definitions that both tools are able to parse, but only Yosys is able to properly synthesize it. This shows that vendor lock-in can go both ways. Looking at the functional designs from Table 1.1 we can see that the **SHA256** is a problem for both tools at a relative error of 75 % for our and 90 % for the vendor estimate. We double-checked this result to ensure no measurement error. **ChaCha** looks more inline with 143 % of relative error for our solution and 109 % for the vendor's.

	Vendor			Ours		
	Picorv32	ChaCha	SHA256	Picorv32	ChaCha	SHA256
SB_LUT4	52435	3734	3044	4282	3728	3066
Total Power [mW]	-	6.9	6	31.6	5.6	9.5
Measured Power [mW]	-	3.3	64	11.8	2.3	38.4
Relative error [%]	-	109	90	167	143	75

Table 1.1: Hardware utilization and power estimation of the practical benchmarks with vendor tool and our solution.

1.5 Community Service

When working on parts of the implementation of this thesis, we encountered a few problems with open source components that we are using. We investigated those issues and submitted fixes upstream.

1.5.1 nextpnr

Using cells with no output worked fine when placing and routing **DFF**s, but the ice40 packing code of nextpnr made it crash when doing the same for **LUT**. The reason for this is that the code that tries packing **LUT**s and **DFF**s into the same logic cell is expecting the **LUT** to have an output. This output is then used to check, if a **DFF** is used in the same logic cell. Due to our benchmarking **LUT** not having an output exposed, the code crashed right at the access of the missing output. The offending code is found in Listing 1.1. The solution to this issue is to check for the existence of the entry at **id_O** and only execute the rest of the **DFF** packing code if it does.

The fix for this bug is submitted as a pull-request[2] and merged into the nextpnr repository.

```
1 NetInfo *o = ci->ports.at(id_O);
```

Listing 1.1: Crashing nextpnr code

[2] https://github.com/YosysHQ/nextpnr/pull/944

1.5.2 yosys

When attempting to simulate ring oscillator circuits with path delays in Icarus Verilog (iverilog) utilizing the simulation library from Yosys revealed that the path delay definitions in said library do not comply to the Verilog standard. This resulted in errors when parsing them in iverilog. This is fixed by our pull request[3] to Yosys.

[3] https://github.com/YosysHQ/yosys/pull/3542

Chapter 2

State of the Art

Commercial vendors provide software tools for power estimation. These tools are quite often locked behind licenses and one vendor can provide different tiers of features depending on the FPGA architecture specific tooling. The tools we investigate in the feature matrix below are the ones included in **Lattice Diamond** and **iCEcube2**, **Altera PowerPlay** and the **Xilinx Power Estimator**, which covers a big portion of around 90% of the FPGA market[1]. Main focus is on Lattice and Xilinx provided tools, due to them being our test platforms.

■…Full-Support
▒…Partial-Support
□…No-Support

	Setup								Output														
	Temperature				Device				Temp.		Power										Visualization		
	Ambient Temp.	Board Spec	Heat Sink	Airflow	Package	Performance Grade	Worst Case	Voltage Source	Junction Temp.	Max. Safe Ambient	General	Per Voltage Source	Per General Block	Per Element	Per Signal	Per Data	General IO	Per IO	Clock Domain	Power Graphs	Bar Charts	Tree List	Text output
Lattice iCEcube2	■						■	▒	■		■						■		■				
Lattice Diamond	■	▒	▒	■	■	■	■	■	■	■	■	■	■				■	■	■	■			■
Intel PowerPlay	■	▒	■	■	■	■	■	■	■	■	■	■	■		■				■				■
Xilinx Power Estimator	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		■	■	■
(our solution) IcePwrEst								■			■	▒	■	■	■	▒	■	■	▒				■

Table 2.1: Feature matrix of vendor tools

Standard procedure for power estimation in literature is to provide a model that gets parameterized by regression algorithms applied on given empirical power data. This data can on the one hand be acquired directly by measurements on hardware [3], which requires elaborate measurement setups to extract the power data. The decent accuracy of the measurements are also important, which is difficult to achieve on highly clocked systems like MCUs/CPUs [7]. On the other hand, measurement data is acquired by utilizing existing vendor power estimation tools [8]. Depending on the features of each tool, these directly provide the power of different components. This second approach requires trust in the accuracy of the vendor provided tools, which can vary to a substantial degree as mentioned in [3] (referencing [2] and [4]). Typically the logic components are building blocks like adders and multipliers [8, 5]. Clock frequencies and the placement and routing are also affecting the dynamic power of an estimate [3]. To estimate the power, activation rates/frequencies also need to be known either by means of estimation or simulation. Estimated values can change based

[1] https://web.archive.org/web/20220125223413/https://seekingalpha.com/article/4481365-lattice-semiconductor-strong-competition-limits-stock-price-upside

on assumptions like zero-delay, logic-delay and interconnect-delay, which introduce additional glitching, due to all this the process of estimating the activation rates is a highly complex task as well [1].

Our main target in this work is the Lattice iCEBreaker FPGA board, which contains an FPGA of the iCE40 UltraPlus family. This family embeds Programmable Logic Blocks (PLB), Embedded Block RAM (EBR), Single Port RAM (SPRAM), etc.... The PLB contains Look-up-Tables with 4 inputs (LUT4), D-Flip-Flops (DFF) with configurable set/reset and enable controls, and carry logic. All literature that we found is working on Xilinx FPGAs investigating power estimation, but the Lattice iCE40 family seems like it hasn't been focus of such research.

Here follows a short summary of the contents of the references that we are basing our work on.

2.1 Summaries

The goal of [1] is to estimate the switching activity of a design and the interconnect capacitance. They report that the power consumption of a Xilinx Virtex II FPGA is to $50 - 70\%$ due to interconnect loss. Switching activity data is generated by simulation of benchmark designs with random input vectors. It compares zero-, logic- and routed-delay transition count, where they show that the amount of transitions increases between each delay category. Then they fit an estimation model onto the routed switching activity and compare the results against the simulation. They show that the zero- and logic-delay switching activity is always underestimating compared to the routed delay activity as the ground truth. Their predictions on the other hand over- and underestimate compared to routed delay activity. Which they claim allows for better results, due to the mispredictions averaging out for the most part. For the interconnect capacitance estimation they generate placed netlists of the various benchmarks and modify the netlist to result in different routing solutions with otherwise identical circuits. This allows them to extract the difference in interconnect capacitance by running the resulting circuits through the Xilinx power estimator. With this they can fit their multi-dimensional model by means of regression analysis to provide estimates for the capacitance. This is a very well made paper, but they only compare to Xilinx power estimator and no hardware measurements. Hardware measurements would have made their contribution even more valuable. For our work we are using their idea of extracting the routed-delay switching activity from simulation.

In [3] they provide a hardware measurement setup for FPGA and techniques to separate the power consumption of the hardware into different groups (static, clock, interconnect and logic power). This enables designers to target the most relevant parts when optimizing for power. Their measurement setup consists of an FPGA, a fixed voltage regulator, a shunt resistor and a differential probe. A DSP controller is used for the input vector stimulus. First they propose a procedure to extract the static and clock power of a given circuit by different input vectors. Then they utilize specifically designed circuits to extract the interconnect power by moving modules inside the FPGA away from the IO-pins. The input buffer capacitance is also extracted by measuring 2 different test circuits. A tool for extraction of wire and interconnect lengths of a design is proposed for Xilinx based FPGAs. At the end they compare their hardware measurements to Xilinx Power Estimator (XPower), which show a significant difference. We used their hardware measurement setup as a basis for our setup. The authors' investigation of XPower's estimation accuracy was also very motivating to not blindly trust vendor tools.

A design flow to consider power utilization at a high design level is proposed in [2]. They show that classic power estimation flows are not efficient, since they have to be redone for every new design. Their approach describes a system

where an application is partitioned in IP-blocks and those blocks are picked from a library with constraint optimized version. If an IP-block does not exist, it is designed, optimized for the constraints and then added to the library. A simple power estimation concept is proposed based on this block scheme. This estimation is compared to real measurements and the XPower tool with considerably better results for their solution than the vendor tool.

Paper [5] describes a methodology to estimate the power of a hardware design by characterizing essential operator blocks. They fit adder and multiplier block and also describe logic, clock and signal dependent power that they model. For activation rate they also provide an estimation model. They provide a hardware measurement setup similar to [3] consisting of the FPGA, a voltage regulator, a shunt and an oscilloscope. The input stimulus is generated by a Spartan3 board. With this they compare their estimate to XPower and hardware measurements.

Reference [8] provides a power model that they fit by using XPower as the ground truth. This model is then compared to a similar previous art and XPower. An interesting anecdote is their use of K-means clustering on their circuits to cluster them into useful groups.

The work in [4] proposes a switched capacitor based measurement method to enable the accurate measurement of low power designs. They compare their setup to another hardware measurement done with a multi-meter and to XPower. Their solution provides decent accuracy with similar results to the multi-meter and that the results from XPower. The setup is even capable of characterizing the state machine of an SDRAM controller. This work is interesting for us, if we needed a high accuracy and high time resolution measurement setup.

In [7] a similar approach to [4] is taken for measurements. They provide a capacitor based measurement setup for small power consumption. Though this design requires more components and special parts like supercapacitors. They validate their setup against a multi-meter on multiple benchmark, where it fairs quite well. The approach is interesting, but for our purposes the more simple switched capacitor solution from [4] is more useful.

Chapter 3

Methodology and Implementation

This chapter goes over the entire implementation of each component that is done in this work, starting with our measurement setup, which is essential for data acquisition. After this, we go over our methodology for designing the testbenches that we are using with our benchmarks. The testing is accompanied by the benchmark designs that we put on the FPGA for evaluation of the hardware. Then we set up a model fitting scheme, which uses the measurements to produce the needed parameters for the estimator. Finally, we use the fitted estimator model on a handful of use case designs to evaluate its performance to the vendor tool.

3.1 Overview

This work is an attempt to create a power estimation tool for a given FPGA architecture. Here we will coarsely go over the methods that we are using in the following parts of the work.

Power measurements are conducted by inserting a shunt resistor in the V_{core} supply line of the FPGA on the evaluation PCB and measuring its voltage drop to get the current, which is proportional to the used power.

Testbenches for stimulation of benchmarks that require multiple changing inputs we resorted to LFSRs generated by a microcontroller. For single input designs we directly map the 12 MHz clock input as the signal. We also implement these testbenches on the simulation side to extract valuable information on internal transitions.

Benchmarks to characterize the FPGA are designed with low complexity in mind to allow for better reasoning about their internal behavior and to reduce the need for complex testbenches. This also allows better debugging of suspicious results. The benchmarks include LFSR, ring oscillator, divider and arbiter based circuits.

Modelling of the FPGA is done as a linear system based on a Complementary Metal-Oxide Semiconductor (CMOS) model (Equations (3.1) to (3.3)) for the components inside the FPGA.

$$P_{\text{t}} = P_{\text{s}} + P_{\text{d}} \tag{3.1}$$

$$P_{\text{s}} = U I_{cell} \tag{3.2}$$

$$P_{\mathrm{d}} = fC_{cell}U^2 \tag{3.3}$$

Model fitting The fitting of the model is done by solving the linear system of the aggregated benchmarks with classic optimization algorithms.

Estimation and evaluation For the assertion of estimation quality we compare the output of our estimator to the vendor tool in three practical use cases, such as a CPU, a stream cipher (**ChaCha**) and a hash algorithm (**SHA256**). Hardware measurements have also been taken to provide a ground truth.

3.2 Power measurements

This section goes over our power measurement setup based on the physical FPGA board, shunt resistors, a current sense amplifier and an oscilloscope. This setup is needed to measure the power utilization of our target hardware.

3.2.1 Initial measurement attempt

Our initial measurement setup is using an USB measurement card with a 12 Bit Analog-Digital Converter (ADC). At a supply voltage of 5 V, this results in at best $\approx 1.2\,\mathrm{mV}$ of quantization step size. From this the minimal current that can possibly be measured from the $1.5\,\Omega$ shunt resistor is $\approx 800\,\mu\mathrm{A}$. This turns out to be insufficient measurement precision, due to the low power FPGA used. Even bigger design targets like the Dhrystone benchmark[1] running on a PicoRV32 core[2] on the iCEBreaker board show power utilization in the range of a few quantization steps, which made those measurements also not very expressive.

These problems mean that we have to put more focus on the measuring setup and benchmarking. Our target for useful measurements is around a minimum of $100\,\mu\mathrm{V}$, which would allow currents in the range of $66\,\mu\mathrm{A}$. This seems to be possible to achieve by amplifying the differential voltage of the shunt with an operational amplifier or a dedicated Integrated Circuit (IC) and measuring the single ended resulting voltage with an accurate ADC. For this work we pursue a solution with a dedicated IC, because such chips have better component matching and less offset errors in addition to very good noise and frequency characteristics.

3.2.2 Current Sense Amplifier Board

To ease the measurement of small differential voltages on the shunt resistors, the current sense amplifier IC INA293B5 is utilized. This IC is a good compromise on gain, bandwidth and noise for our purposes. Its most interesting electrical characteristics are listed in Table 3.1 Figure 3.1 shows an amplifier circuit that supports 3 differential inputs. The bill of materials is found in Table 3.2.

3.2.3 iCEBreaker FPGA Measurement Setup

The iCEBreaker board is using a low power Lattice iCE40UP5k FPGA IC, which has a small amount of logic cells compared to Xilinx FPGAs. This results in the situation that even the power draw of designs that fill almost the entire FPGA, is very minor.

[1] https://github.com/YosysHQ/picorv32/tree/master/dhrystone
[2] https://github.com/YosysHQ/picorv32

	INA293B5
Gain [V/V]	500
PSRR [μV/V]	0.1
CMRR [dB]	140
Bandwidth [kHz]	900
Voltage noise density [nV/$\sqrt{\text{Hz}}$]	50

Table 3.1: Electrical characteristics of the INA293B5

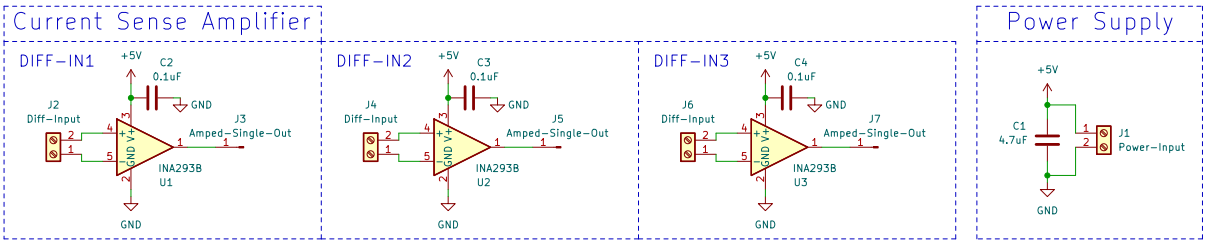


Figure 3.1: Schematic of the amplification circuit for 3 differential inputs

Component	Value	Reference	Count
OpAmp	INA293B5	U1, U2, U3	3
Capacitor	0.1 μF	C2, C3, C4	3
Capacitor	4.7 μF	C1	1
	Screw Terminals	J1, J2, J4, J6	4
	Pins	J3, J5, J7	3

Table 3.2: Bill of material for amplifier board

The iCEBreaker PCB has preexisting jumper pads shown in Figure 3.2, which are by design intended to be used for shunt resistors to allow for current measurements of the FPGA. For our setup 1.5 Ω shunt resistors are used. The setup to measure the power of the board is depicted in Figure 3.3. The differential voltage of the V_{core} shunt resistor is getting amplified by the current sense amplifier PCB Section 3.2.2. The IO shunt resistors on the other hand are connected to subtraction circuits due to higher currents, which clip when amplifying with the INA293B. The outputs of all amplifiers are measured with an oscilloscope.

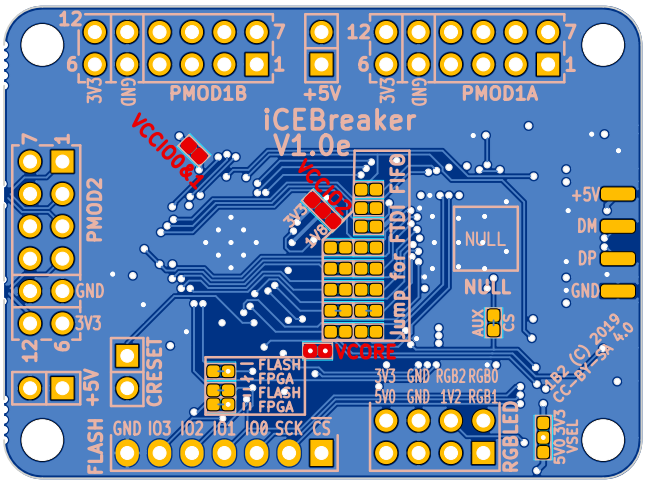


Figure 3.2: Bottom side of iCEBreaker PCB with position for measurement shunts marked in red

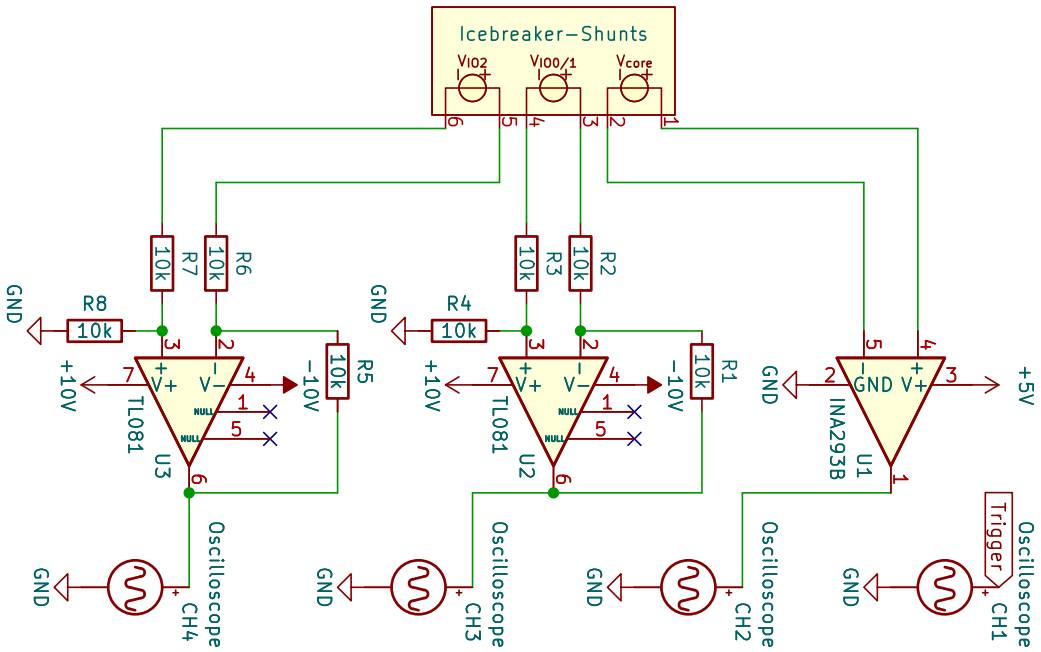


Figure 3.3: Schematic of the power measurement setup on the iCEBreaker board

3.2.4 Hardware Measurement

This section contains initial measurements based on the validation setup Figure 3.4. At first, we validate the measurement setup with a simple test with known parameters. After this we conduct initial power measurements on the iCEBreaker board with our first microbenchmarks to gain some insight into the FPGA's behavior.

Amplifier Prototype Validation

Before the amplifier board is used for target hardware measurements it needs validation to confirm it properly working. The simple validation setup is depicted in Figure 3.4. The variable values in Table 3.3 are selected in such a way that a current of approximately $0.1\,\mathrm{mA}$ is measured at the shunt resistor R_{SH} resulting in $1\,\mathrm{mV}$. If the circuit is working correctly, this differential voltage should get amplified by the INA293 to around $0.5\,\mathrm{V}$.

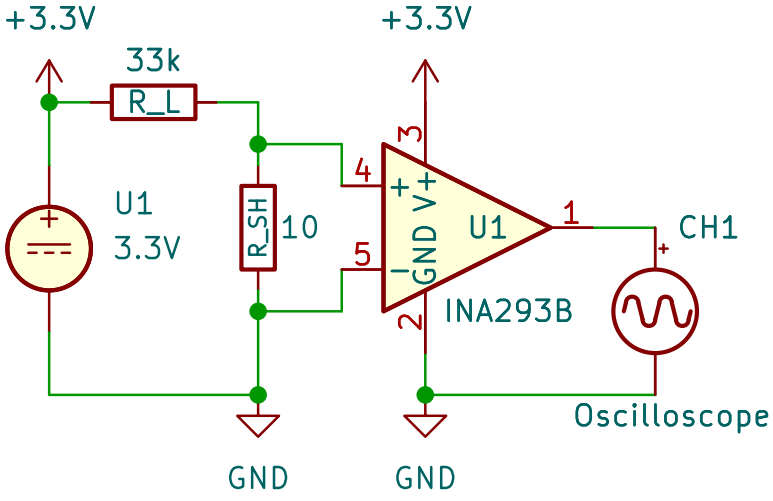


Figure 3.4: Schematic of the amplifier prototype validation setup

	picked value	measured value
$U1\,[\mathrm{V}]$	3.3	3.3
$G\,[\mathrm{V/V}]$	500	–
$R_{\mathrm{L}}\,[\mathrm{k\Omega}]$	33	31.8
$R_{\mathrm{SH}}\,[\Omega]$	10	10.3

Table 3.3: Component values of the validation setup

Measurement Results

For the measurements it has to be taken into account that the resistance of the connecting points lies in the range of $0.1 \ldots 0.7\,\Omega$, which includes the wire and contact resistance. The resistor tolerances used in the setup are $1\,\%$. The three amplifiers are measured one after another. The measurement results of the differential voltage U_{diff}, the amplified output voltage U_{out} and the calculated amplified voltage $U_{\mathrm{amp\_calc}}$ are found in Table 3.4. The relative errors of $1.2 \ldots 6.25\,\%$ between the amplified and calculated values are acceptable when comparing to the uncertainties and variances of the whole setup.

The overall behavior of the circuit is correct, which validates its functionality for this DC test. This provides the needed basis for utilizing the circuit for power measurements of the FPGA board.

	Amp1	Amp2	Amp3
$U_{\mathrm{out}}\,[\mathrm{V}]$	0.415	0.417	0.416
$U_{\mathrm{diff}}\,[\mathrm{mV}]$	0.82	0.8	0.78
$U_{\mathrm{amp\_calc}}\,[\mathrm{V}]$	0.41	0.4	0.39
$e_{\mathrm{rel}}\,[\%]$	1.20	4.08	6.25

Table 3.4: Measurements from the validation setup

3.3 Testbench

This section explains the method used to generate stochastic inputs and how they are used to test designs. Testbenches are important to explore and test the behavior of designs. In our case we utilize them as a reproducible stimulus for our target benchmark designs. We go over the basic design of our testbench, the Verilog implementation and the hardware realization of our setup.

3.3.1 LFSR based testing

To stimulate the input of a DUT with stochastic inputs that are not too highly correlated a pseudo random sequence generated by an LFSR is used.

This testing method is simple to describe for reproduction, because the LFSR has only 3 degrees of freedom. These are the polynomial order, the feedback taps and the initial register seed. In practice these degrees of freedom are reduced even more, when only polynomials with maximum length are chosen. This limits the sets of feedback taps that are used for a given polynomial order. Lists of polynomial orders and their maximum length tap configurations are found in most literature about LFSRs and also on the Internet[3].

[3] http://users.ece.cmu.edu/~koopman/lfsr/

3.3.2 Verilog Testbench

A Verilog testbench is utilized to simulate the DUT with the stochastic inputs that are used in the hardware testbenches. The simulation of the testbench is used to acquire an approximate representation of the internal signals of a design running on hardware. It is not fully accurate, since no gate or interconnect delays are taken into account. The values are rather a lower bound of the internal signal changes. These signal changes are used to calculate activation rates, which are required for most power estimation models.

3.3.3 Microcontroller Testbench

A microcontroller testbench is used to examine a design on hardware. It is based on a microcontroller that is connected to the FPGA IOs and the measurement device. The FPGA contains the DUT design which gets stimulated by stochastic signals from the microcontroller. The measurement device receives a trigger signal from the microcontroller when the testing begins. The microcontroller itself is controlled by the PC, which starts the test runs and is able to configure some testing parameters. The FPGA is also connected to the PC to allow for exchange of the DUT. The data from the measurement device is either directly copied to the PC or indirectly via a data storage depending on the capabilities of the device.

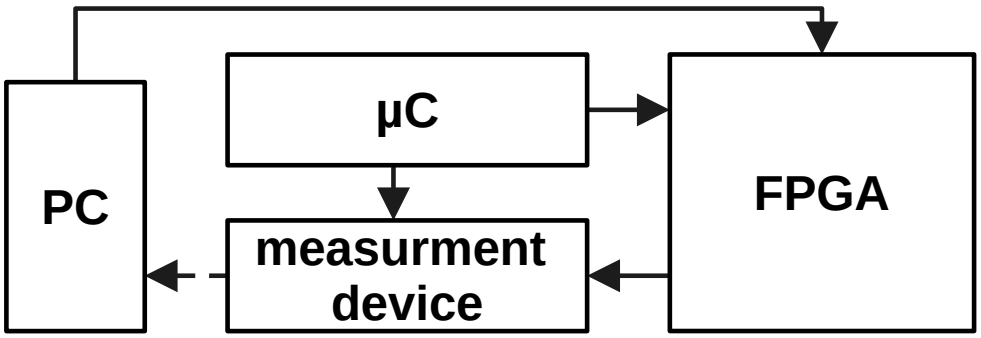


Figure 3.5: Principle of microcontroller testbench with measurement device

3.4 Benchmarks

Benchmarks are essential for targeted analysis of certain aspects of FPGA hardware. These benchmarks are tedious to create and vary in parameters like component count or placement position. Thus we decided to create a simplified way of creating such microbenchmarks by providing a tool, which generates said benchmarks in Verilog and an accompanying constraint file from a benchmark definition file. The microbenchmarks depend on the target toolchain and hardware, which requires specific handling in the generator tool. In particular we are looking into Lattice iCE40 FPGAs, due to their well supported open source workflow.

Our main goal is to accelerate the creation of microbenchmarks for FPGA hardware analysis by providing a simple tool that generates said benchmarks and its variations. The resulting Verilog files are then used by other tools also utilizing the definition file to implement testbenches, which conduct simulations and automated hardware testing.

After this we also present benchmarks that use the **generate** statement from Verilog to create parametrizable designs that can very flexibly scale in FPGA resource usage.

3.4.1 Generator

The basic idea is to instantiate a minimum amount of cells and move them around in the FPGA. This includes not connecting the output of the component to reduce wires. Such outputless components are detected by the optimization tool. To prevent them being optimized away it is important to add the Verilog **keep** attribute.

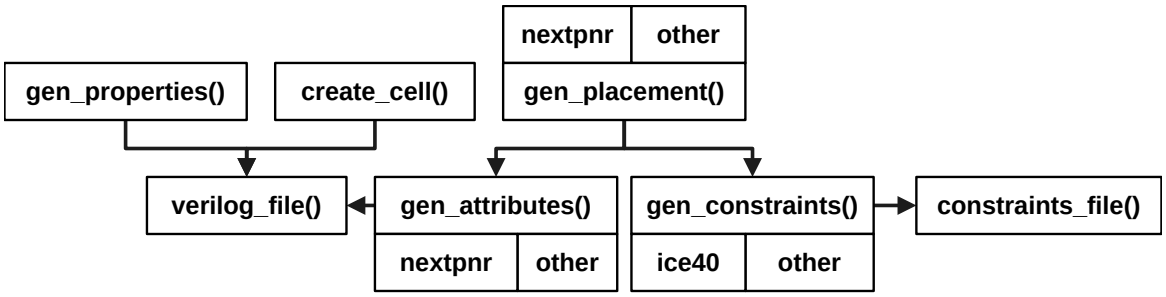


Figure 3.6: Benchmark generator hierarchy and data flow

The various toolchain vendors require different handling of the placement constraints of components. Figure 3.6 shows the general hierarchy of the functions of the benchmark generator and their data dependencies. Two examples in the next sections require completely different parts of the pipeline to override placement.

3.4.2 Nextpnr (Yosys)

Nextpnr is a place and route tool that in combination with the synthesis tool Yosys supports the entire hardware synthesis process for the Lattice iCE40 FPGA family. To forcefully place a component with Nextpnr one has to specify the attribute in Listing 3.1. For example a look-up-table is positioned by specifying its X and Y coordinate on the grid of the FPGA and the desired logic cell. The Lattice iCE40up5k has for example 8 logic cells per valid grid position.

```
1 (* BEL="X4/Y4/lc3" *)
```

Listing 3.1: Verilog attribute to place a component into the given logic cell

3.4.3 iCEBreaker FPGA – Evaluation

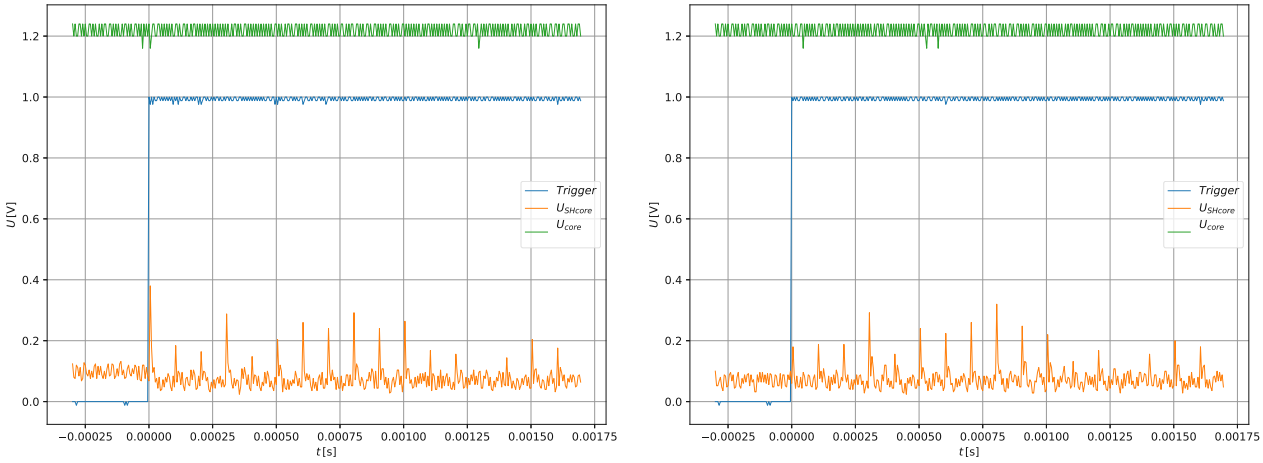
The measurements of the Lattice iCE40UP5k FPGA are of particular interest to us for analyzing the internals of the FPGA. To gather information about the properties of internals of the FPGA, different circuits and measurements are used. The following sections go over the different circuits and their measurement results.

LUT4

One of the basic internal building blocks of the FPGA is the LUT with 4 inputs and one output, which is measured in this section. The circuit in this setup consists of LUT only, that get instantiated by the benchmark generator in Section 3.4.1. The inputs of all LUTs are connected to 4 inputs of the FPGA in parallel. This allows to control all LUTs by only using 4 inputs. The idea behind connecting all LUTs in parallel is to be able to measure their power in an additive way. This should result in a proportional increase of power in regards to the number of LUTs used. For example reducing the LUT amount from 5 k to 1 k would result in approximately a fifth of the power after correcting for a constant offset value.

Figure 3.7 contains measurements of the iCEBreaker board with 5 k and 1 k LUT instantiated on the FPGA respectively. These inputs then get stimulated by an maximum length 4Bit-LFSR every 100 μs after the trigger signal rising. When looking at Figure 3.7 one can see the voltage spikes in a regular pattern. This pattern matches the LFSR changing values and one can also see its period of 15. Comparing the three graphs does not bring the desired result of proportional dependency of the power to the number of LUT. This means the voltage spikes on the shunt are the result of something else. For further investigation Figure 3.8 is created. The setup for these measurements is one instantiated LUT and only one input connected to one bit of the LFSR. The purpose of this measurement is to investigate how the power relates to the connected signals and their state.

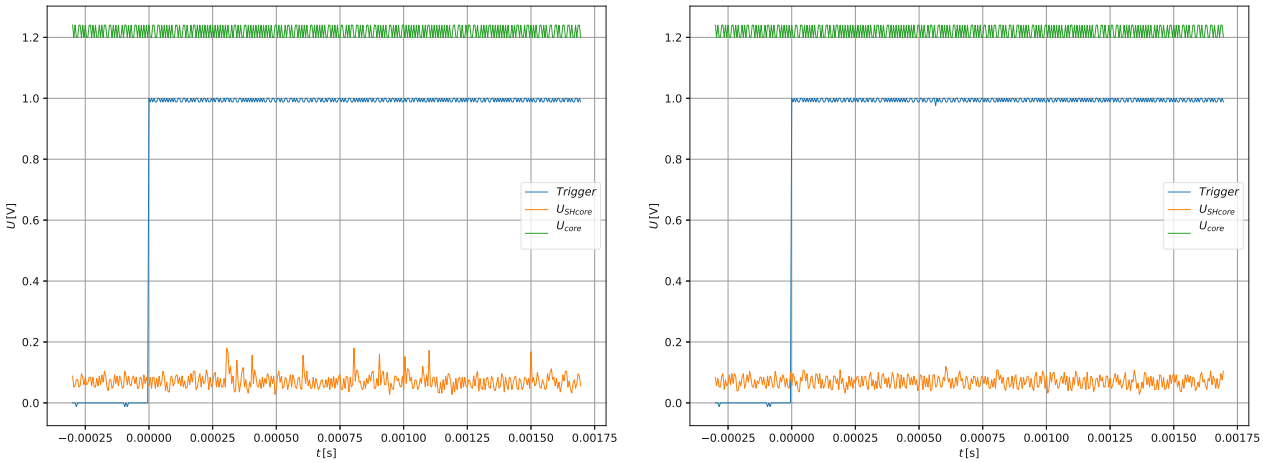
Comparing the measurements from Figure 3.7 and Figure 3.8 shows that the voltage on the shunt resistor does not depend on the number of LUT at all, due to there being no real difference in the amplitude of the voltage spike. On the other hand the comparison allows the conclusion that the amplitude of the voltage spike depends on the amount of inputs in high state. In Figure 3.8 the voltage spikes occur only sparsely due to the LFSR still being of period 15 and only one bit being connected to the LUT.



(a) 5 k LUT4

(b) 1 k LUT4

Figure 3.7: Measurements of LUTs stimulated by 4Bit-LFSR



(a) stimulated by 1 output of 4Bit-LFSR

(b) not stimulated

Figure 3.8: Measurements of 1 LUT4

```
1  module ringoscillator_test_not(
2      input [0:0] test_in1,
3      output [0:0]test_out1
4  );
5
6  wire internal_w[5:0];
7  wire last_w[0:0];
8
9  assign test_out1[0] = last_w[0];
10 (* keep *)
11 and(internal_w[0],test_in1[0],last_w[0]); //and(Y,A,B) Y = A&B
12 (* keep *)
13 not(internal_w[1],internal_w[0]); //not(Y,A) Y = ~A
14 (* keep *)
15 not(internal_w[2],internal_w[1]);
16 (* keep *)
17 not(internal_w[3],internal_w[2]);
18 (* keep *)
19 not(internal_w[4],internal_w[3]);
20 (* keep *)
21 not(internal_w[5],internal_w[4]);
22 assign last_w[0] = internal_w[5];
23 endmodule
```

Listing 3.2: Example code of Verilog high level ringoscillator implementation

This testing shows that this type of benchmarks is not sufficient to analyze low power FPGA hardware like the Lattice iCE40UP5k chip on the iCEBreaker FPGA board. Filling the FPGA hardware to the brim with simple constructs and stimulating it at low frequency is not enough to get measurable power consumption.

Ring-oscillator

The next circuit to test for power measurements is a simple ring oscillator. This is constructed by daisy-chaining an uneven number of **NOT**-gates together and connecting the output of the last gate with the input of the first. Such a construct is used on FPGAs for example to generate random numbers.

As a first attempt to instantiate a ring oscillator the Verilog code in Listing 3.2 is used. This code should generate a ring oscillator that is turned on and off with an FPGA input via an **AND**-gate. The **keep** attribute prevents the synthesis tool from optimizing gates away.

After a few synthesis steps in Yosys, the resulting circuit in Figure 3.9 shows that the attempt of using **keep** is futile and that most gates are optimized away. The reason for the optimization kicking in is due to the translation of the **NOT** and **AND**-gates to LUT, which the **keep** attribute did not propagate to. This means that a more elaborate way to instantiate the ring oscillator is required. It is also important to note that in the place and route step the additional flag **–ignore-loops** has to be used for nextpnr to allow the placement of such an oscillator. This in turn disables the static timing analysis of the router.

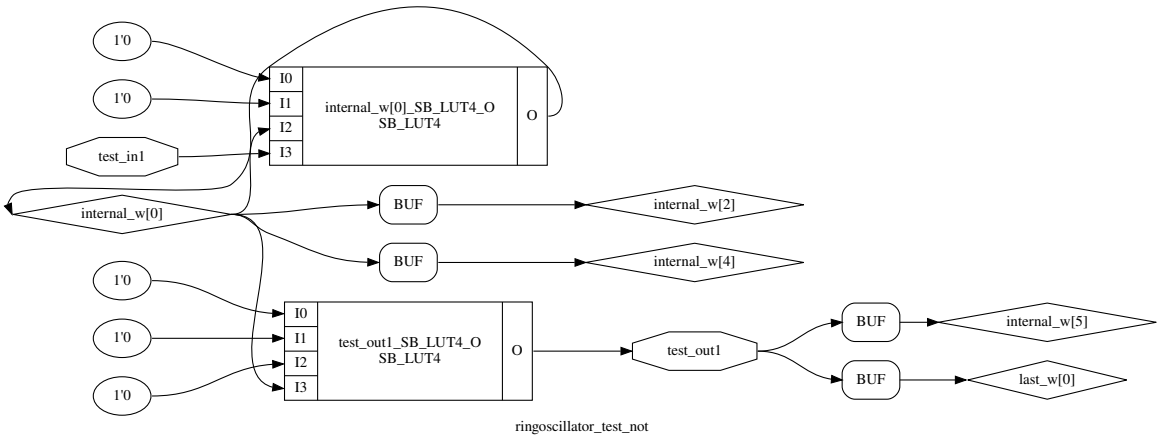


Figure 3.9: Schematic of the synthesized **not** based ring oscillator

```
module ringoscillator_test(
    input test_in1,
    output test_out1
);

parameter [15:0] not_cfg= 16'b0000_0000_0000_0001;
parameter [15:0] and_cfg= 16'b0000_0000_0000_1000;

wire internal_w[5:0];
wire last_w[0:0];

assign test_out1 = last_w[0];

(* keep  *) (* BEL="X1/Y1/lc0" *)
SB_LUT4 #(.LUT_INIT(and_cfg)) test_lut_x1y1bel0(.I0(test_in1),.I1(last_w[0]),.I2(1'b0),.I3(1'b0),.O(internal_w[0])); //and
(* keep  *) (* BEL="X1/Y1/lc1" *)
SB_LUT4 #(.LUT_INIT(not_cfg)) test_lut_x1y1bel1(.I0(internal_w[0]),.I1(1'b0),.I2(1'b0),.I3(1'b0),.O(internal_w[1])); //not
(* keep  *) (* BEL="X1/Y1/lc2" *)
SB_LUT4 #(.LUT_INIT(not_cfg)) test_lut_x1y1bel2(.I0(internal_w[1]),.I1(1'b0),.I2(1'b0),.I3(1'b0),.O(internal_w[2])); //not
(* keep  *) (* BEL="X1/Y1/lc3" *)
SB_LUT4 #(.LUT_INIT(not_cfg)) test_lut_x1y1bel3(.I0(internal_w[2]),.I1(1'b0),.I2(1'b0),.I3(1'b0),.O(internal_w[3])); //not
(* keep  *) (* BEL="X1/Y1/lc4" *)
SB_LUT4 #(.LUT_INIT(not_cfg)) test_lut_x1y1bel4(.I0(internal_w[3]),.I1(1'b0),.I2(1'b0),.I3(1'b0),.O(internal_w[4])); //not
(* keep  *) (* BEL="X1/Y1/lc5" *)
SB_LUT4 #(.LUT_INIT(not_cfg)) test_lut_x1y1bel5(.I0(internal_w[4]),.I1(1'b0),.I2(1'b0),.I3(1'b0),.O(internal_w[5])); //not
assign last_w[0] = internal_w[5];
endmodule
```

Listing 3.3: Example code of Verilog low level ringoscillator implementation

Utilizing the tricks from the benchmark generator we can approach the issue from further down the synthesis flow. The code in Listing 3.3 shows a functionally equivalent implementation of the ring oscillator as described in Listing 3.2, but this time the general gates are replaced with the iCE40 specific LUT. Figure 3.10 show that this time the synthesis tool did not optimize the intermediate gates away. This means with this approach we can generate the desired ring oscillators that we want to measure on FPGA hardware.

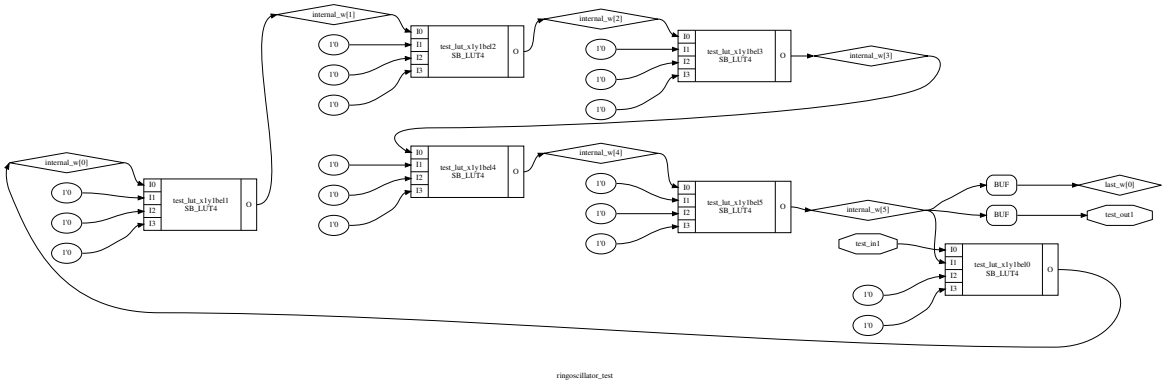
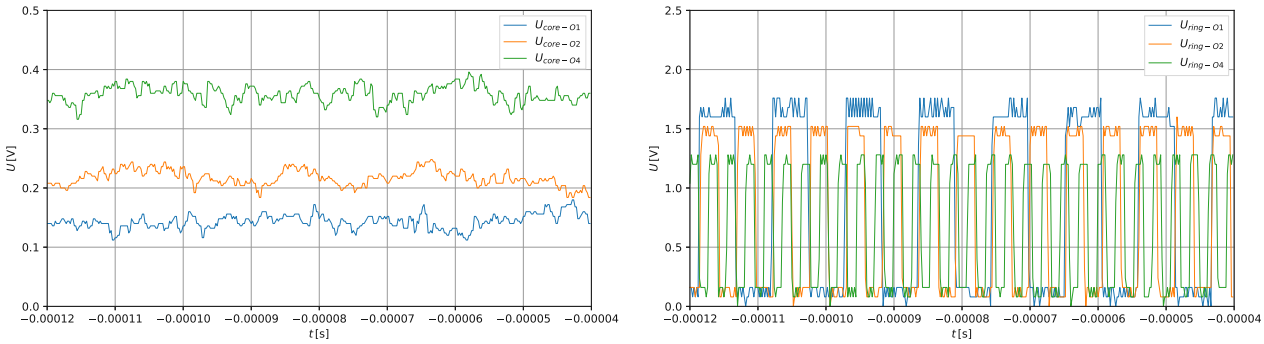


Figure 3.10: Schematic of the synthesized LUT based ring oscillator

For the following results, a total number of approximately 3000 **NOT** gates is instantiated on the iCEBreaker board. These gates are all chained together in the same fashion as shown in Listing 3.3 for the first ring oscillator $O1$. The second design $O2$ consists of two chains of half the length of $O1$ with double the oscillation frequency. The third the design $O4$ consists of half the length of $O2$ with double the chains, which results in two times the frequency. This scaling is useful to evaluate the sensitivity of power usage to frequency changes.

Figure 3.11b shows the output of the oscillating signal of one chain in each of the three designs. One can clearly observe the doubling of the oscillator frequency between each of the designs by counting the number of positive edges between the period of the previous oscillator. This confirms the oscillator designs properly working and scaling. The significant measurements are found in Figure 3.11a, which shows an increasing voltage on the V_{core} shunt that corresponds to the average core current. Using the measurement and assuming an approximately constant core voltage it is trivial to calculate the dynamic power used based on the simple model from Equation (3.4).



(a) Differential voltage on the V_{core} shunt resistor of three different ring oscillators

(b) Outputs of the three ring oscillators used

Figure 3.11: Measurements of 3000 NOT gates in a ring oscillator configuration

```
1  module not_mod(output b, input a);
2  assign b=~a;
3  endmodule
4
5  module ringoscillator_test_not(
6      input [0:0] test_in1,
7      output [0:0]test_out1
8  );
9
10 wire internal_w[5:0];
11 wire last_w[0:0];
12
13 assign test_out1[0] = last_w[0];
14
15 and (internal_w[0],test_in1[0],last_w[0]); //and(Y,A,B) Y = A&B
16
17 (* keep_hierarchy *)
18 not_mod not1(internal_w[1],internal_w[0]); //not(Y,A) Y = ~A
19
20 (* keep_hierarchy *)
21 not_mod not2(internal_w[2],internal_w[1]);
22
23 (* keep_hierarchy *)
24 not_mod not3(internal_w[3],internal_w[2]);
25
26 assign last_w[0] = internal_w[3];
27 endmodule
```

Listing 3.4: Example code of Verilog high level ring oscillator implementation

Post-experimental improvement discovery

After further research on Verilog attributes we found the perfect fit that we can use for easier creation of the ring oscillator circuits. At this point the experiments have already been conducted, but the improved construct is still useful for the following benchmarks. The attribute in question is **keep_hierarchy**. This prevents the in-lining of modules in the flatten synthesis step by keeping their hierarchical boundary. This means that optimizations will only affect the connecting circuits and the low-level implementation, but not the interface boundary itself.

Figure 3.12 shows a ring oscillator with 3 **NOT** gates and a **AND** to turn the oscillator on and off. This design is synthesized down to the iCE40 technology level from Listing 3.4. The **NOT** module contains only one **SB_LUT4**, which results in 4 LUT used for the entire oscillator.

Divider – Setup

Dividers are useful to manipulate the existing clocks in a controlled manner. Benchmarking them allows usage in other benchmarks. As a divider design we settle on dividing by powers of two, since this is realized by a counter and accessing the n-th bit for the divided output. Listing 3.5 contains the source code that we use.

We test a few different combinations of dividers as seen in Table 3.5. The parameters that get varied are the number of instances **NUM** and the division factor **DIV**. The 12 MHz clock of the FPGA is the input of the dividers. The actual

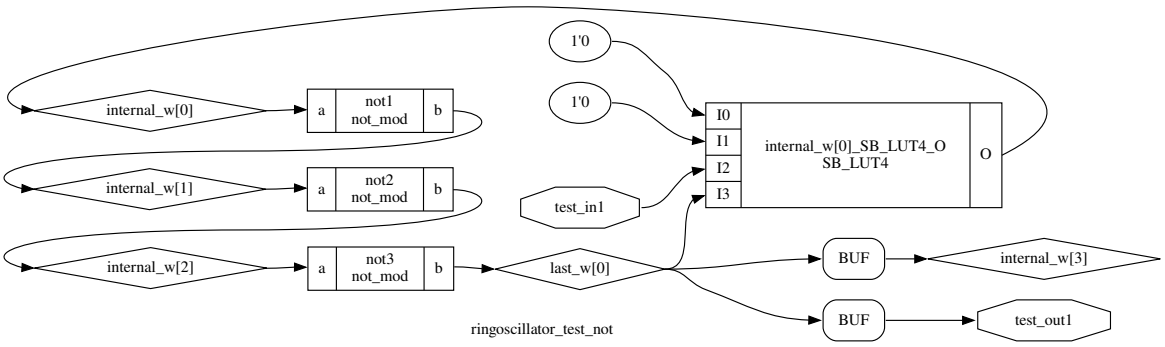


Figure 3.12: Schematic of the synthesized **not** based ring oscillator

```
module div
#( parameter integer DIVIDER = 5)
(   input clk,
    output d);

reg [DIVIDER:0] cnt = 0;

always @ (posedge clk)
begin
    cnt <= cnt+1'b1;
end

assign d = cnt[DIVIDER];

endmodule
```

Listing 3.5: Example code of Verilog high level divider implementation

measurement evaluation is done in Section 3.4.4.

DIV	1	2	4	5	6	8
NUM	50	50	50	1	1	1
NUM	100	100	100	50	50	50
NUM	150	150	150	100	100	100
NUM	250	250	250	250	150	150
NUM	500	500	500	500	250	250
NUM	2000				500	

Table 3.5: All measured divider configurations.

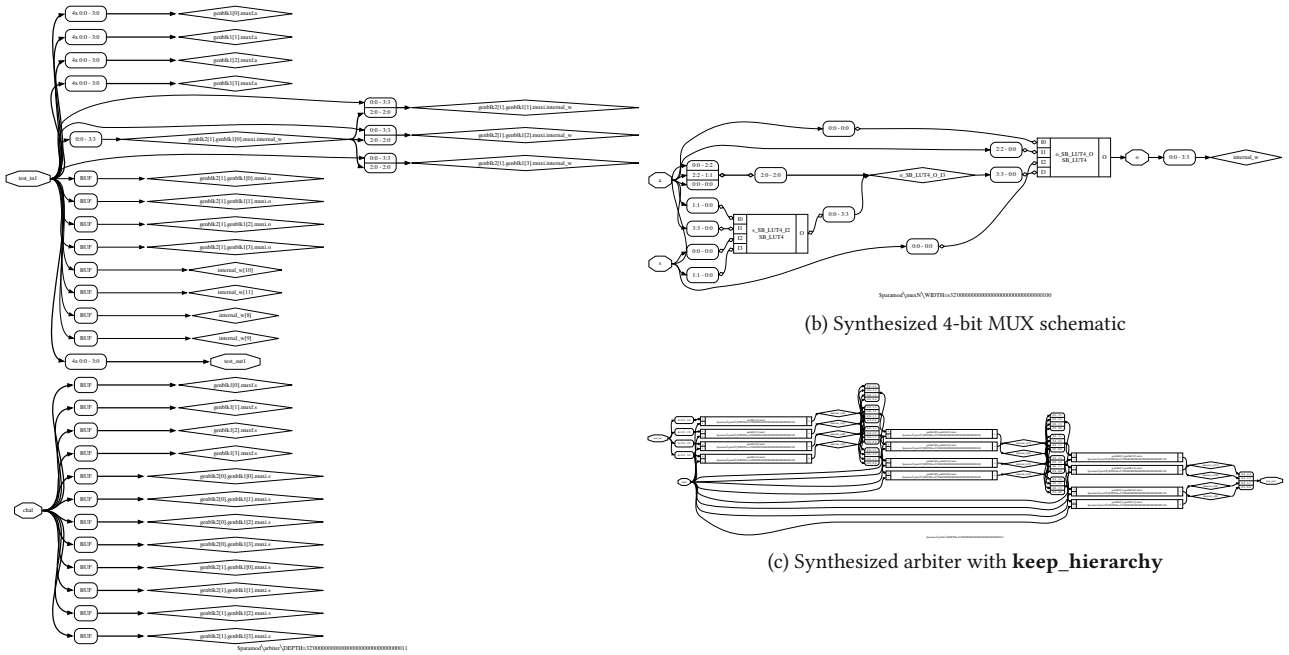
Arbiter – Setup

Our arbiter consists of **WIDTH** wide multiplexer (MUX) with the same count in parallel and **DEPTH** amount of them daisy-chained together. The MUX (Figure 3.13b) is implemented with a variable width to allow for a more configurable arbiter. Listing 3.6 contains the Verilog code for the arbiter on which the schematics are based upon. It can freely be configured by the corresponding Verilog properties. A helpful Yosys feature is the command **attrmap** to disable the **keep_hierarchy** attribute. This allows the use of a single Verilog file to generate the comparison shots in Figure 3.13. In Figure 3.13a one can observe a synthesized arbiter without the **keep_hierarchy** attribute, which gets optimized away completely. Figure 3.13c on the other hand shows the proper version.

To create a useful benchmark out of the arbiter we add a divider as the input. The divider has the 12 MHz clock of the FPGA as an input, which is then divided by powers of 2 with the parameter **DIV** representing the exponent of the division factor. The different combinations of configurations in Table 3.6 are measured. The actual measurement evaluation is done in Section 3.4.4.

```
module muxN
#( parameter integer WIDTH = 4)
(input [WIDTH-1:0] a, input [$clog2(WIDTH)-1:0] s, output o);
genvar i;
generate
wire [WIDTH-1:0] internal_w;
assign internal_w[0] = a[0] & (s == 0);
for (i=1; i<WIDTH; i=i+1) begin
    assign internal_w[i] = (a[i] & (s == i)) | internal_w[i-1];
end
endgenerate
assign o = internal_w[WIDTH-1];
endmodule

module arbiter
#( parameter integer DEPTH = 600, //600 => 4792LUT4, 300 => 2100LUT4
   parameter integer WIDTH = 4)
(   input test_in1,
    input [$clog2(WIDTH)-1:0] chal, //input [$clog2(WIDTH)*DEPTH-1:0] chal,
    output [WIDTH-1:0] test_out1);
localparam integer GATE_CNT=2*1500;

genvar j;
genvar i;
generate

wire internal_w[DEPTH-1:0][WIDTH-1:0];
wire challenge_w[DEPTH-1:0][WIDTH-1:0];
//connect inputs of arbiter
for (i=0; i<WIDTH; i=i+1) begin
    (* keep_hierarchy *)
    muxN #(.WIDTH(WIDTH)) muxf({test_in1, test_in1, test_in1, test_in1}, chal[($clog2(WIDTH)-1):0], internal_w[0][i]);
end

// setup remaining mux connections
for (j=0; j<DEPTH-1; j=j+1) begin
    for (i=0; i<WIDTH; i=i+1) begin
        (* keep_hierarchy *) muxN #(.WIDTH(WIDTH)) muxi({internal_w[j][(i+3)%WIDTH], internal_w[j][(i+2)%WIDTH],internal_w[j][(i+1)%WIDTH], internal_w[j][(i+0)%
            WIDTH]},
        chal[($clog2(WIDTH)-1):0], internal_w[j+1][i]);//chal[(($clog2(WIDTH))*(j+2)-1):(($clog2(WIDTH))*(j+1))]
    end
end

//connect outpus of arbiter
for (i=0; i<WIDTH; i=i+1) begin
    assign test_out1[i] = internal_w[DEPTH-1][i];
end

endgenerate
endmodule
```

Listing 3.6: Example code of Verilog high level arbiter implementation



(a) Synthesized arbiter without **keep_hierarchy**

(b) Synthesized 4-bit MUX schematic

(c) Synthesized arbiter with **keep_hierarchy**

Figure 3.13: Arbiter synthesis comparison

LFSR – Setup

An LFSR is interesting as it is a simple structure with pseudo-random behavior. It is also used to stimulate other systems if a seemingly random and wide input is needed. Similarly to the arbiter circuit the LFSR is also stimulated by the FPGA 12 MHz clock going through a divider. The **NUM** parameter is the number of parallel instances of the LFSR. The actual

DEPTH	100	100	100	100	300	300	300	300	600	600	600	600	600
WIDTH	4	4	4	4	4	4	4	4	4	4	4	4	4
DIV	1	2	4	8	1	2	4	6	1	1	2	5	8

Table 3.6: All measured arbiter configurations.

measurement evaluation is done in Section 3.4.4.

DIV	0	1	2	4	8
NUM	100	100	–	100	100
NUM	400	400	400	–	400
NUM	–	800	800	800	800

Table 3.7: All measured LFSR configurations.

3.4.4 Combined Power Measurement – Results

In this section we present the power measurements of all the circuits shown previously. The reason for this is that all measurements share the same hypothesis of scaling power with utilization and stimulation frequency.

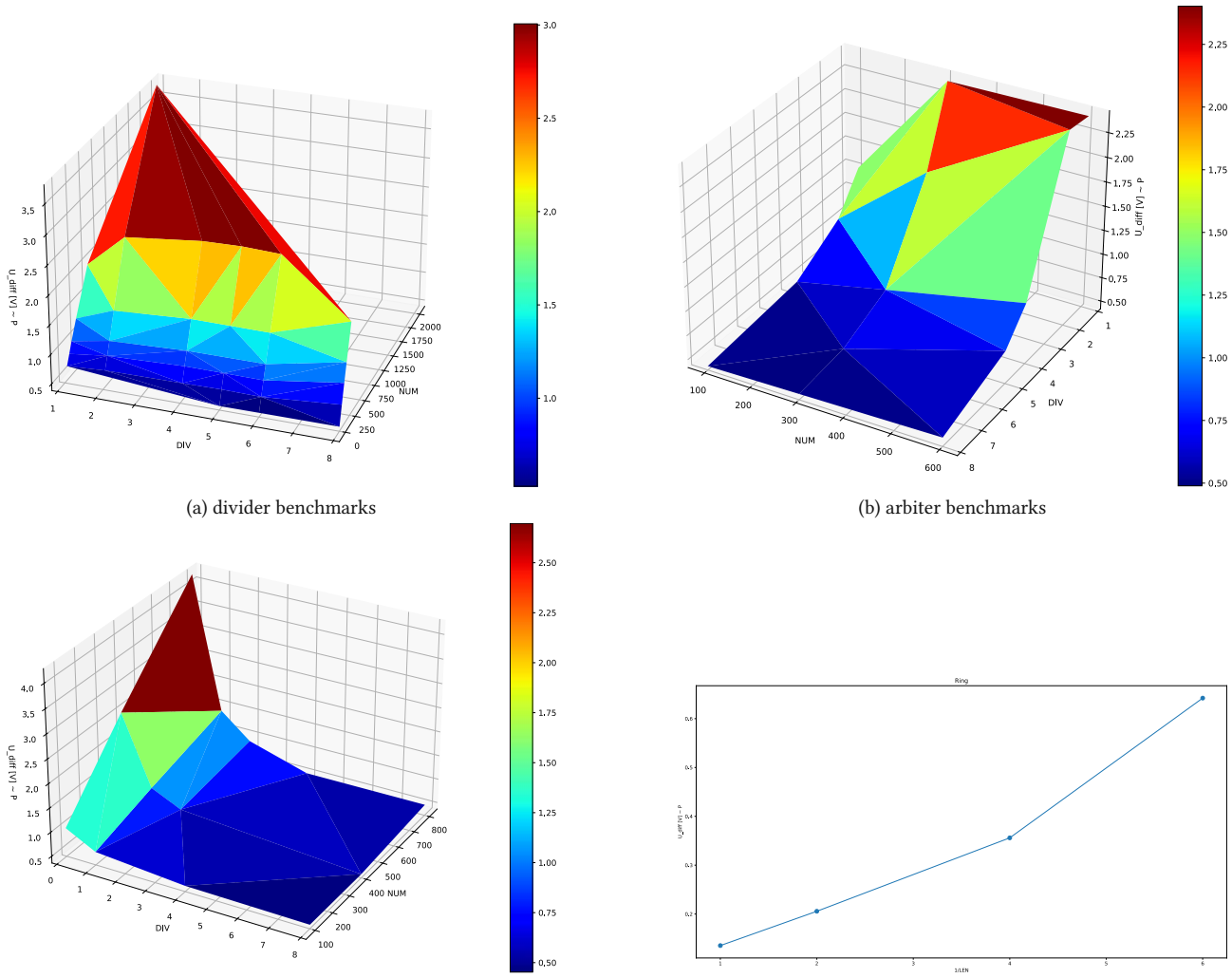
The 3-dimensional plots Figure 3.14a, 3.14b and 3.14c plot the parameter space in the X-Y-plane and the power utilization on the Z-axis. The graphs all show the same trend of increasing power when reducing the division factor **DIV** or increasing the number of instances **NUM**. The ring oscillator in Figure 3.14d only has one parameter **1/LEN**, since the number of instances is kept constant. This means with each length reduction one identical parallel ring oscillator is instantiated to make up for the reduced instances in one oscillator chain. The main takeaway point from Figure 3.14 is to show that the power draw of each benchmark scales as intended with changing its parameters. Increasing the number of instances, decreasing the division factor or reducing the length all should result in higher power draw. On some measurement this is not the case, but this is very likely due to measurement inaccuracies from the setup.

With the confirmation of the expected power scaling of our benchmarks we can move on to building a model for power estimation in Section 3.5 and fitting it with our gathered data.

3.4.5 iverilog iCE40 simulation with delays

Having a working simulation environment to work with is useful to evaluate the behavior of designs beforehand. This is especially important for low level implementations on FPGA logic blocks and high frequency circuits like ring oscillators. For the low-level implementation a zero delay simulation is good enough, but the high frequency circuits require approximate timings of the FPGA. A high frequency ring oscillator can cause the destruction of the FPGA by overloading the internal paths due to high frequency switching. Synthesis tools might detect such high frequency paths and prevent them, but designers can choose to implement the oscillator by disabling these safeguards. To prevent the destruction it is important to know the approximate oscillation frequency, which is determined from simulation that includes the gate delays. The frequency is an upper bound to the real frequency, due to path delays being missing in this step.

For Verilog simulation the open source tool iverilog is used. Verilog that is synthesized down to the iCE40 technology mapping is simulated with iverilog by including the simulation library from Yosys. It is to note the Verilog path delay definitions in Yosys's simulation library do not comply to the Verilog standard and iverilog errors out when parsing



(a) divider benchmarks

(b) arbiter benchmarks

(c) LFSR benchmarks

(d) ring oscillator benchmarks

Figure 3.14: Combined power measurements of all benchmarks.

them. This is fixed by our pull request[4] to the library.

Listing 3.7 shows an example on how to simulate an iCE40 synthesized design with iverilog. The argument **-gspecify** is necessary to enable the usage of Verilog **specify** blocks, which contain the path delay definitions. With the flag **-D** the corresponding Verilog define variables are set. Yosys's simulation library contains timing definitions for the different variants of the iCE40 architecture. To select the low power variant the variable **'ICE40_LP=1'** has to be defined.

```
iverilog \
        -gspecify \
        -D 'VCDFILE="simulation_trace.vcd"' \
        -D 'NO_ICE40_DEFAULT_ASSIGNMENTS=1' \
        -D 'ICE40_LP=1' \
        -o simulation_output \
        $(yosys-config --datdir/ice40/cells_sim.v)) \
        test_ice40.v
vvp simulation_output
```

Listing 3.7: Example code to run a simulation including path delays with iverilog

[4] https://github.com/YosysHQ/yosys/pull/3542

3.5 Model Fitting

After the heavy lifting of designing benchmarks and acquiring data from the hardware in Section 3.4.3, this chapter goes over fitting this data to a simple model. Figure 3.15 shows the conceptual flow of the entire analysis of a single benchmark **benchmark.v** plus its testbench **tb.v**. Most parts of the analysis flow didn't deserve their own section, so they are packed together in Table 3.8 with a short description. The design analyzer Section 3.5.1 and model fitting Octave script Section 3.5.2 on the other hand are the key ingredient, specifically created to accomplish our power estimation goal. The following parts will go over each step in the analysis flow.

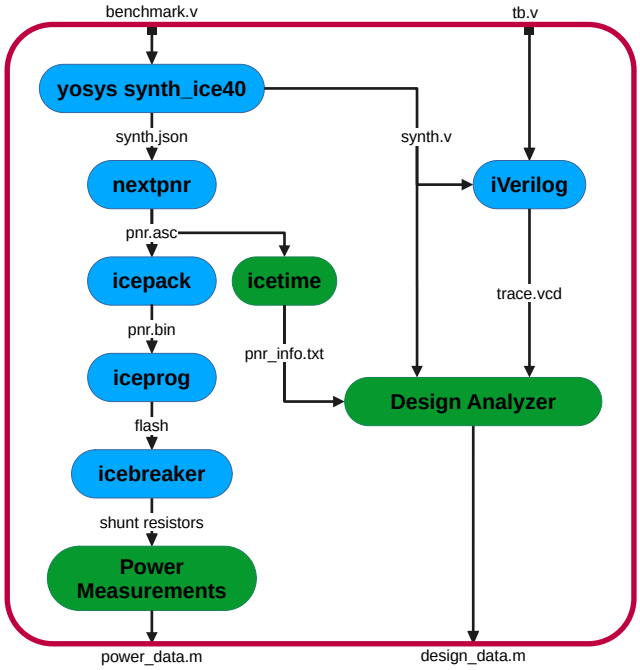


Figure 3.15: Data flow diagram of the analysis flow. ■...Unmodified Tool, ■...Adapted or created Tools

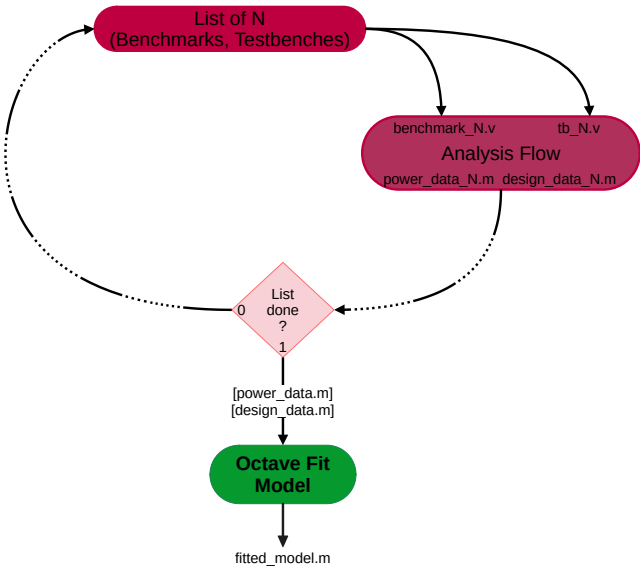


Figure 3.16: Flow diagram of the entire flow to calculate the fitting parameters from a list of benchmarks.

3.5.1 Design Analyzer

Design Analyzer is our main tool to parse previously generated files, containing design specific properties, **synth.v**, **trace.vcd** and **pnr_info.txt**. This data is then combined into per-cell entries in **design_data.m**.

In the first step the synthesized Verilog file **synth.v** is parsed into a map of all module instances and their name. The module instances are differentiated between parent modules and the hardware-architecture-specific cells (LUT, DFF, CARRY). After that the VCD file **trace.vcd** is parsed into a map of the signal names and their toggles, metadata and the delta time in that the signal toggles occur in. Combining all the previous data the Verilog modules get an activation frequency assigned to their ports. As a final step the data structure is dumped into **design_data.m**, a file that Octave is able to parse. Containing a utilization matrix $\boldsymbol{M}$ where each row describes the type of cell represented by a module as shown in Table 3.9 and $\boldsymbol{f}$ containing the sum of the activation frequencies of all ports of a module.

3.5.2 Octave – Fit Model

An FPGA consists of basic cells like LUT, **DFF**, **CARRY**, etc. These cells are made from CMOS logic. The classic way to model the power of CMOS logic is to consider a static term (Equation (3.5)) and a dynamic term (Equation (3.6)), which

Yosys – Synthesis	Yosys takes the benchmark design **benchmark.v** and synthesizes it down to a target FPGA architecture specific netlist in the form of **synth.json** and a Verilog design file **synth.v**.
iVerilog – Simulation	iVerilog is used to simulate the synthesized benchmark **synth.v** in combination with the corresponding testbench **tb.v**. This produces a trace file **trace.vcd** containing information about how all the components of the synthesized design react to the testbench stimulus.
nextpnr – Place and Route	Nextpnr utilizes the netlist generated by Yosys and places it onto the given FPGA device structure. The placed cells are then connected by routing their connections with the wires and MUXes provided by the FPGA resulting in the **ASCII** encoded bitstream file **pnr.asc**.
icepack – Packing	Icepack simply packs the human-readable bitstream from **pnr.asc** into a binary representation **pnr.bin**, which is needed for the flash memory on the FPGA.
iceprog – Flashing	Iceprog flashes the bitstream **pnr.bin** via USB onto the iCEBreaker FPGA board.
icebreaker – Hardware	The iCEBreaker board is configured via the bitstream **pnr.bin** and provides shut resistors to measure its current.
Power Measurements	The power measurements are done via hooking of the shunt resistors as described in Section 3.4.3. The measurements are preprocessed as **power_data.m** into the matrix format that Octave can read to make them directly usable in later steps.
Icetime – Timing Data	Icetime is a static timing analysis tool for the iCE40 FPGA, which is used by nextpnr to determine if frequency constraints are meet with a given bitstream or not. We modified it to be able to export all internal timing information of each cell in **pnr_info.txt**.

Table 3.8: Description of analysis flow parts

LUT	$1, 0, 0$
DFF	$0, 1, 0$
CARRY	$0, 0, 1$
PARENT	$0, 0, 0$

Table 3.9: Rows of ice40 architecture mapping per cell type

are summed in Equation (3.4) to give an approximation of the real power consumption. This means the simplest yet reasonable model for the elemental FPGA cells assuming linearity is to consider them the same as a CMOS gate. The total power consumption of an FPGA is calculated by adding all individual cells together (Equation (3.7)). This results in a power model that is linear when considering the cell properties I_{cell} and C_{cell}. Such a linear model is represented as a product of a matrix and a vector (Equation (3.11)).

$$P_\mathrm{t} = P_\mathrm{s} + P_\mathrm{d} \tag{3.4}$$

$$P_\mathrm{s} = U I_\mathrm{cell} \tag{3.5}$$

$$P_\mathrm{d} = f C_\mathrm{cell} U^2 \tag{3.6}$$

$$P_\mathrm{T} = \sum^{\mathrm{cells}} P_\mathrm{t} \tag{3.7}$$

With a bit of effort the multitude of benchmark measurements and data are arranged into the matrix form of a linear transformation and then solved with an algorithm for linear systems i.e. least squares provided by the computer algebra system. Due to linearity of the underlying model most data from a benchmark is collapsed into lower dimensional vectors

U	supply voltage
f	activation frequency of a cell
C_{cell}	equivalent capacitance of a cell
I_{cell}	static leakage current of a cell
P_s	static power dissipation of a cell
P_d	dynamic power dissipation of a cell
P_t	total power dissipation of a cell
P_T	total power dissipation of a system of cells
$\boldsymbol{P_{\mathrm{t,B}}}$	vector of P_T entries from benchmarks
$\boldsymbol{M}$	cell utilization matrix of a design
$\boldsymbol{M_{ji}}$	scalar of the utilization matrix at row j and column i
s_i	aggregated static utilization of column i
$\boldsymbol{s}$	static usage aggregation vector of a design
$\boldsymbol{S}$	static usage aggregation matrix from benchmarks
$\boldsymbol{f}$	activation frequency vector of a design
$\boldsymbol{F}$	activation frequency matrix of a design
f_i	accumulated activation frequency of column i
$\boldsymbol{f}_\mathrm{acc}$	frequency accumulation vector of a design
$\boldsymbol{F}_\mathrm{B}$	frequency accumulation matrix from benchmarks
$\boldsymbol{c}$	parasitic capacitance of cell types vector
$\boldsymbol{i}$	static current of cell types vector
$dim(\boldsymbol{P}_{\mathrm{t,B}}) = q$	number of benchmarks used
$dim(\boldsymbol{x}) = n$	number of variables
$dim(\boldsymbol{A}) = q \times n$	
o	number of used cells of a benchmark
$dim(\boldsymbol{F}) = o \times n$	

Table 3.10: Explanation of all the variables used in the fitting model equations.

as by Equation (3.16) and Equation (3.12). This greatly reduces the size of the data vectors of each benchmark.

Starting off with a single benchmark one can calculate the total power of the system over all its cells with Equation (3.8). It is to note that cells of the same type are collapsed into one entry, so this sum over cells only means unique cells. After this one can arrange multiple total power values of benchmarks into a sum of vectors (Equations (3.9) to (3.10)). The sums in these vectors is rewritten as vector products $\mathbf{Si}$ and $\boldsymbol{F}_\mathrm{B}\mathbf{c}$. $\mathbf{S}$ is the result of the column sum of the utilization matrix $\boldsymbol{M}$ as in Equations (3.12) to (3.14), which describes the amount of cells with static leakage current utilization. $\boldsymbol{F}_\mathrm{B}$ describes the accumulated activation frequency of each benchmark as per Equations (3.15) to (3.18). The vector $\mathbf{i}$ contains the static leakage current of each cell type in its entries and $\mathbf{c}$ the parasitic capacitance per cell type. As a final step, the sum of the two vectors is repacked into a linear transformation in Equation (3.11), which concludes the preparations that have to be done to the system to solve it easily with common solver algorithms.

$$P_\mathrm{T} = \sum^{\text{cells}} P_\mathrm{s} + P_\mathrm{d} \tag{3.8}$$

$$\boldsymbol{P}_{\mathrm{t,B}} = [P_{\mathrm{T},i} \dots P_{\mathrm{T},q}]^\intercal = \begin{bmatrix} \sum^{\text{cells},i} P_{\mathrm{s},i} + P_{\mathrm{d},i} \\ \vdots \\ \sum^{\text{cells},q} P_{\mathrm{s},q} + P_{\mathrm{d},q} \end{bmatrix} = \begin{bmatrix} \sum^{\text{cells},i} P_{\mathrm{s},i} \\ \vdots \\ \sum^{\text{cells},q} P_{\mathrm{s},q} \end{bmatrix} + \begin{bmatrix} \sum^{\text{cells},i} P_{\mathrm{d},i} \\ \vdots \\ \sum^{\text{cells},q} P_{\mathrm{d},q} \end{bmatrix} = \tag{3.9}$$

$$= \begin{bmatrix} \sum^{\text{cells},i} UI_\mathrm{cell} \\ \vdots \\ \sum^{\text{cells},q} UI_\mathrm{cell} \end{bmatrix} + \begin{bmatrix} \sum^{\text{cells},i} fC_\mathrm{cell}U^2 \\ \vdots \\ \sum^{\text{cells},q} fC_\mathrm{cell}U^2 \end{bmatrix} = U \begin{bmatrix} \sum^{\text{cells},i} I_\mathrm{cell} \\ \vdots \\ \sum^{\text{cells},q} I_\mathrm{cell} \end{bmatrix} + U^2 \begin{bmatrix} \sum^{\text{cells},i} fC_\mathrm{cell} \\ \vdots \\ \sum^{\text{cells},q} fC_\mathrm{cell} \end{bmatrix} = \tag{3.10}$$

$$= \underbrace{U^2 \boldsymbol{F}_{\mathrm{B}} \mathbf{c}}_{\boldsymbol{P}_{\mathrm{d,B}}} + \underbrace{U \mathbf{S} \mathbf{i}}_{\boldsymbol{P}_{\mathrm{s,B}}} = \underbrace{\left[U^2 \boldsymbol{F}_{\mathrm{B}} | U \boldsymbol{S}\right]}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{i} \end{bmatrix}}_{\boldsymbol{x}} \tag{3.11}$$

$$s_i = \sum_{j=1}^{o} \boldsymbol{M}_{ji} \tag{3.12}$$

$$\boldsymbol{s} = [s_1 \dots s_n] \tag{3.13}$$

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s}_1 \\ \vdots \\ \boldsymbol{s}_q \end{bmatrix} \tag{3.14}$$

$$\boldsymbol{F} = \boldsymbol{f}.\boldsymbol{M} \tag{3.15}$$

$$f_i = \sum_{j=1}^{o} \boldsymbol{F}_{ji} \tag{3.16}$$

$$\boldsymbol{f}_{\mathrm{acc}} = [f_1 \dots f_n] \tag{3.17}$$

$$\boldsymbol{F}_{\mathrm{B}} = \begin{bmatrix} \boldsymbol{f}_{\mathrm{acc},1} \\ \vdots \\ \boldsymbol{f}_{\mathrm{acc},q} \end{bmatrix} \tag{3.18}$$

Model Properties and Constraints

Since the linear CMOS model Equation (3.11) is derived from a physical model, it has a few constraints to keep in mind when running fitting algorithms on it or using it. The most important constraint is that the constructed matrix $\boldsymbol{A}^\intercal \boldsymbol{A}$ has a full rank. This means that its rows and columns are linearly independent of all others. As the second property the parameter vector $\boldsymbol{x}$ in our model represents the physical values of capacitance $\boldsymbol{c}$ and leakage current $\boldsymbol{i}$. A capacitance can only have positive values, which constrains $\boldsymbol{c}$ to be always greater or equal to zero. In general, an electrical current can be negative, but the leakage current has to describe a power loss. Hence, the current $\boldsymbol{i}$ also has to be strictly non-negative. A consequence of the non-negativity of both the values in $\boldsymbol{x}$ and $\boldsymbol{A}$ is, that the resulting power estimate can only be non-negative as well. This is important because a negative resulting power estimate would make no sense in a model describing loss.

A summary of the properties:

1. The rank of $\boldsymbol{A}^\intercal \boldsymbol{A}$ is full.
2. The parameters of $\boldsymbol{x}$ is non-negative.
3. The estimated power is non-negative.

3.5.3 Octave Model Fit

Solving the linear model from Section 3.5.2 for its parameters $\boldsymbol{x}$ is done with the least squares approximation (Equation (3.20)). Given the matrix $\boldsymbol{A}$ has full column rank, which implies that the columns of the matrices $\boldsymbol{F}_{\mathrm{B}}$ and $\boldsymbol{S}$ are linearly independent to themselves and each other. To calculate all the required matrices the computer algebra system Octave is used. It takes the two output files form the analysis flow **power_data.m** and **design_data.m** to determine the unknowns of the linear model. The number of columns of the matrix $\boldsymbol{M}$ from **design_data.m** determines the dimension

of $\boldsymbol{x}$ by $2\mathrm{col}(\boldsymbol{M}) = \dim(\boldsymbol{x})$. This means with the current setup a linear system with three cell types has 6 unknowns to solve for. After solving the system one only requires the design specific matrix $\boldsymbol{A}$ to estimate the power utilization of the design. The upcoming chapters go over estimation on the benchmarks used for fitting and some specific use cases that are not in the training set.

$$\boldsymbol{P}_{\mathrm{t,B}} = \boldsymbol{A}\boldsymbol{x} \tag{3.19}$$

$$\boldsymbol{x} = inv(\boldsymbol{A}^{\intercal}\boldsymbol{A})\boldsymbol{A}^{\intercal}\boldsymbol{P}_{\mathrm{t,B}} \tag{3.20}$$

Chapter 4

Results

4.1 Power Estimation of Benchmarks

This chapter covers combining all the former results to generate data and compare it to the hardware power measurements, which is considered our ground truth. Comparing to the real hardware is important, since using the vendor power estimators would not tell us how accurate we are to the real world.

4.1.1 Benchmarks used for fitting

As an immediate comparison, the fitting benchmarks are fed into the power estimation algorithm and are directly compared to the measured data with the CMOS model Equation (4.1). As a bonus model, the static power of the CMOS model is omitted to create a solely dynamic power model (Equation (4.2)). Fitting this extra model came practically for free from the code of the CMOS model. To calculate a power estimate from a given design the entire analysis process used for calculating the fitting parameters has to be run, but at the end instead of the power data the fitted parameters x are used in the model equations. This means due to the fitting process the design characteristics matrix $\boldsymbol{A}$ is already available and directly used on the benchmarks.

$$\boldsymbol{P}_{\mathrm{t,B,CMOS}} = \underbrace{U^2 \boldsymbol{F}_{\mathrm{B}} \mathbf{c}}_{\boldsymbol{P}_{\mathrm{d,B}}} + \underbrace{U \mathbf{Si}}_{\boldsymbol{P}_{\mathrm{s,B}}} = \underbrace{\left[U^2 \boldsymbol{F}_{\mathrm{B}} | U \boldsymbol{S}\right]}_{\boldsymbol{A}_{\mathrm{CMOS}}} \underbrace{\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{i} \end{bmatrix}}_{\boldsymbol{x}_{\mathrm{CMOS}}} \tag{4.1}$$

$$\boldsymbol{P}_{\mathrm{t,B,dyn}} = \underbrace{U^2 \boldsymbol{F}_{\mathrm{B}} \mathbf{c}}_{\boldsymbol{P}_{\mathrm{d,B}}} = \underbrace{\left[U^2 \boldsymbol{F}_{\mathrm{B}}\right]}_{\boldsymbol{A}_{\mathrm{dyn}}} \underbrace{\left[\boldsymbol{c}\right]}_{\boldsymbol{x}_{\mathrm{dyn}}} \tag{4.2}$$

Figure 4.1 shows the bar plot of all 62 benchmarks with three bars per benchmark. First (*blue*) bar in the triple representing the measurement done on hardware, second (*orange*) the estimation result from the CMOS model and last (*green*) the purely dynamic model. In numbers the overall average relative error from the power estimation of the dynamic $e_{\mathrm{avg,rel,P_{dyn}}}$ and CMOS $e_{\mathrm{avg,rel,P_{CMOS}}}$ model is shown in Equation (4.3). The CMOS model reduces the error by $34\,\%$ almost halving it compared to the dynamic model. The overall trend of the CMOS model looks quite promising with over and underestimation on the benchmark designs. The dynamic model on the other hand performs very poorly in

general and is not worth pursuing any further.

Considering the variation seen in Figure 4.2 between some of the benchmarks more model structures have to be explored to possibly find improvement opportunities.

$$e_{\mathrm{avg,rel,P_{dyn}}} = 80.0\,\% \qquad e_{\mathrm{avg,rel,P_{CMOS}}} = 46.1\,\% \tag{4.3}$$

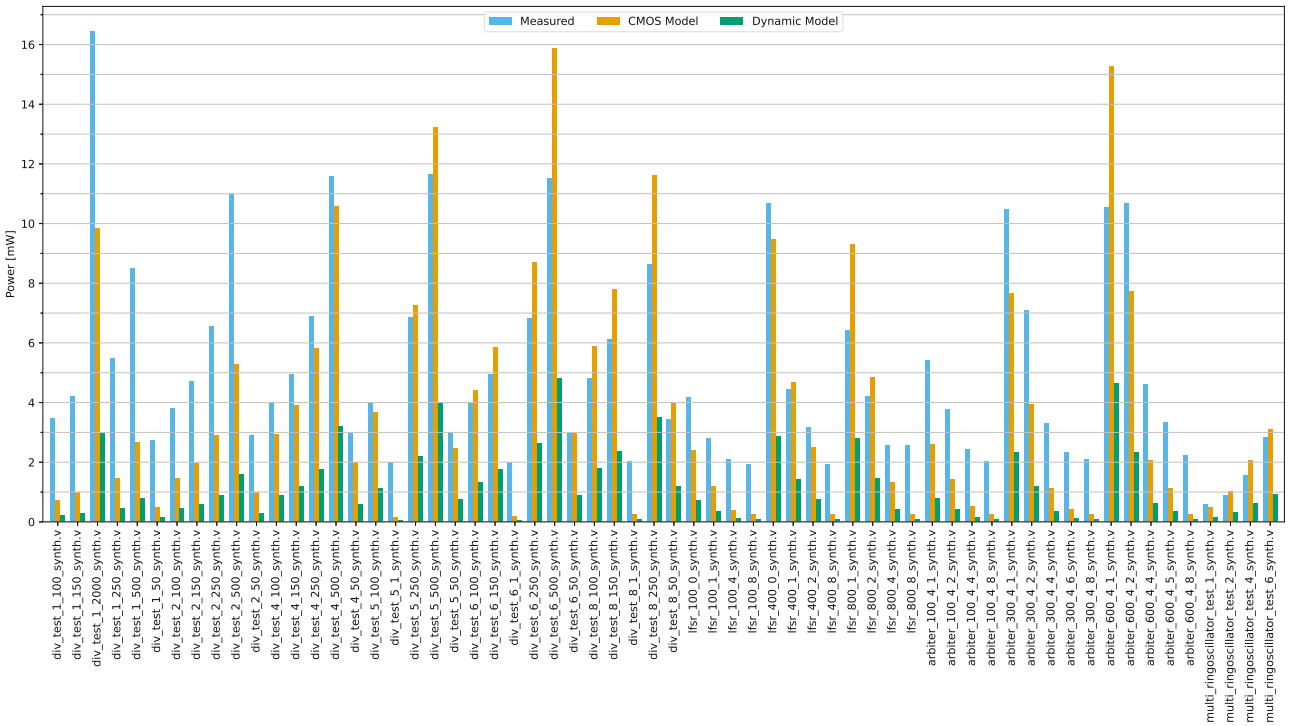


Figure 4.1: Results of running the fitted CMOS and dynamic model over the benchmarks used for fitting.

Per IO CMOS Model Fitting

Going from 3 CMOS elements (combined all IOs) to 12 CMOS elements describing one IO of the fundamental components each shows an improvement across the board in the benchmarks. Especially some outliers like **div_test_5_1**, **div_test_6_1**, **div_test_8_1**, are now better estimated. Other divider benchmarks haven't improved by quadrupling the fitting variables. An investigation of the trace from the **div_test_1_100** showed that the values for the number of instances and the division factor are swapped. This means that i.e. for this benchmark only 1 instance is created and the division factor is 2^{100}. Such a high division factor results in practically no activation rate, which make the benchmarks not useful for our purpose.

4.2 Improvements to the implementation

In this section we go over a few validation and optimization steps some of which are already teased in previous sections.

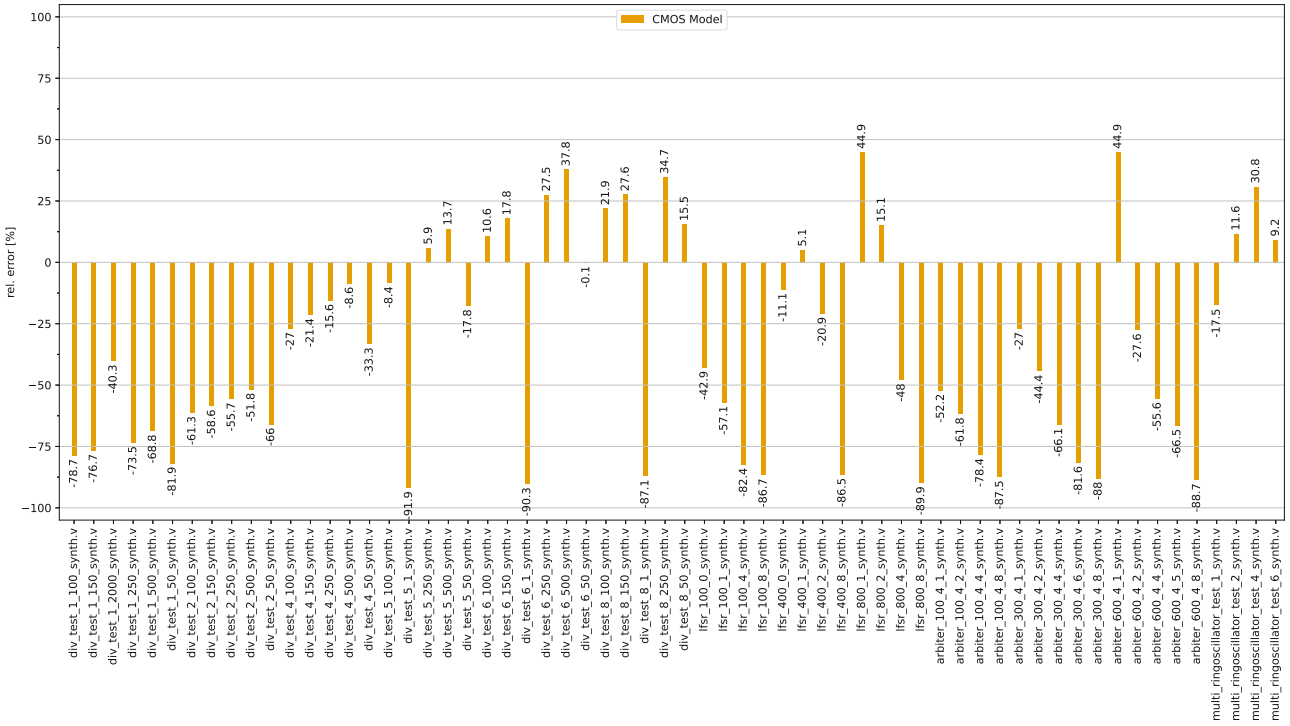


Figure 4.2: Relative error of CMOS model per benchmark estimate to measurement.

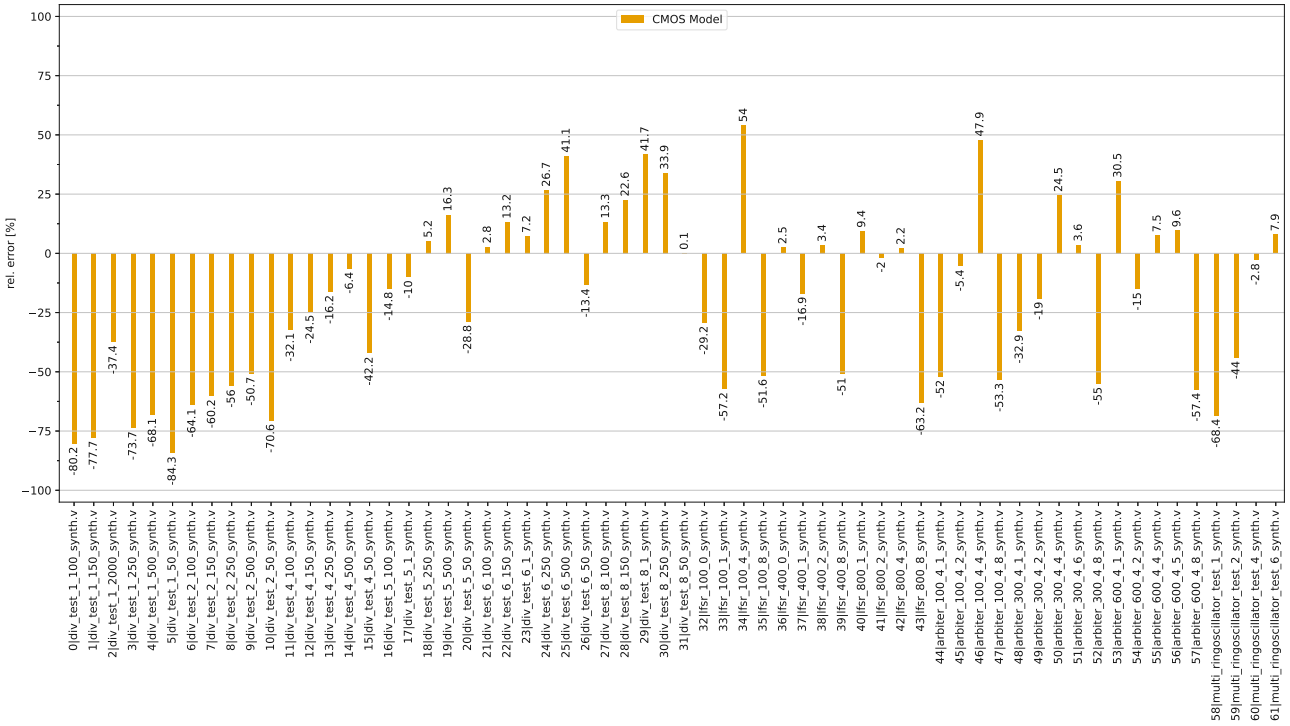


Figure 4.3: Relative error of CMOS model per benchmark with per IO CMOS.

4.2.1 Validation of vector x

Following the previous advice from Section 3.5.2 we analyze the vector x resulting from our least squares solution. It turns out some of the parameters in the fitted x are negative, which is undesirable and needs to be remedied. Since x results from the fitting algorithm used and the input data, we resolve this by changing the fitting algorithm.

Non Negative Least Squares

A robust way to prevent the parameters of vector $\boldsymbol{x}$ to become negative is to solve the given system not as a least squares problem, but instead as an optimization problem with an inequality constraint of $\boldsymbol{x} \geq 0$. Sticking to the idea of the least squares solution the NNLS exists to solve the exact problem we have at our hands. The NNLS is provided as a standard function **lsqnonneg** in Octave and Matlab. Equipped with this tool we can get a solution for $\boldsymbol{x}$, which adheres to our constraints of the physical values represented in the vector. With this the possibility of negative power estimates is remedied as well.

4.2.2 Validation of matrix A

Taking a look at the resulting matrix from $\boldsymbol{A}^\intercal\boldsymbol{A}$ we can calculate its rank, which should be the same as its number of rows/columns. Unfortunately this is not the case for the data generated by our fitting benchmarks.

To solve this issue we can for one get rid of the entries in $\boldsymbol{x}$ corresponding to linear dependent columns in $\boldsymbol{A}^\intercal\boldsymbol{A}$, but this will arbitrarily degenerate our model by getting rid of plausible (from a physics perspective) parameters.

The second knob to adjust are the fitting benchmarks where we can add more diverse benchmarks, which hopefully improve the linear independence of the current parameter set.

Singular Value Decomposition

Some readers with a bit of experience in linear algebra might have noticed by now that due to the rank deficit of $\boldsymbol{A}^\intercal\boldsymbol{A}$ we can not actually solve the least squares problem with the given matrix anyway. So why is Octave able to produce an answer for $\boldsymbol{x}$? Let's zoom in on the bit of code that does all the heavy lifting!

Listing 4.1: Calculation of least squares solution in Octave

```
1   x = pinv(A'*A)*A'*P;
```

As we can see, we are in fact inverting an indefinite matrix, which is not possible! Taking a closer look at the documentation of the inversion function that we are using reveals the following: "*Return the pseudoinverse of x. Singular values less than tol are ignored.*"[1]. So we are not inverting the matrix with a classic algorithm, but instead we find the pseudo inverse by utilizing the Singular Value Decomposition (SVD). This algorithm can handle indefinite matrices by setting problematic singular values to 0, which allows it to return even with seemingly unsolvable inputs. This explains why we get a more or less decently working solution for our least squares problem even with more or less bogus inputs.

Estimates of Practical Benchmarks

This section shows our first comparison of the benchmarks in Section 4.3. For the power estimations we incorporated all discoveries up to this point.

4.2.3 Missing Components

After further analysis of the synthesized components we realize that we miss variations of the DFF in our component handler. These unassigned components certainly contribute to the power consumption, which further adds to cover

[1] https://octave.sourceforge.io/octave/function/pinv.html

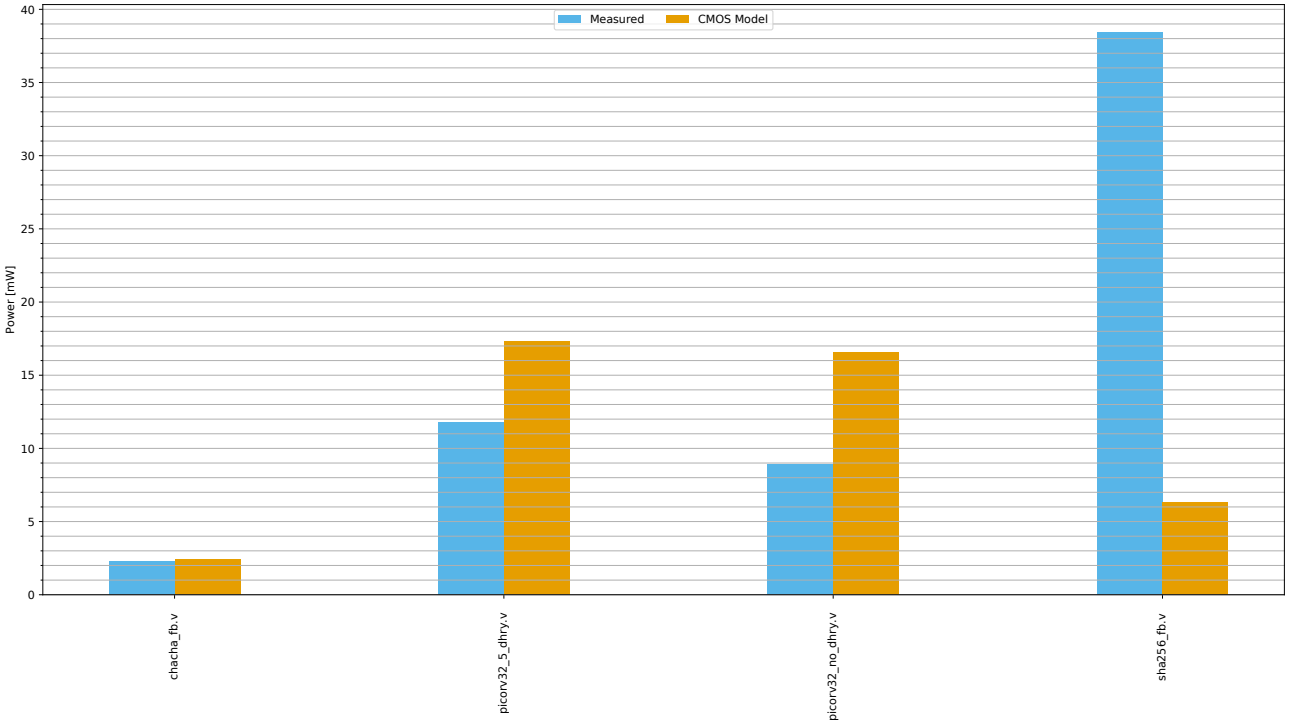


Figure 4.4: Measured and estimated (CMOS model) power of the practical benchmarks with NNLS and all the previews model improvements.

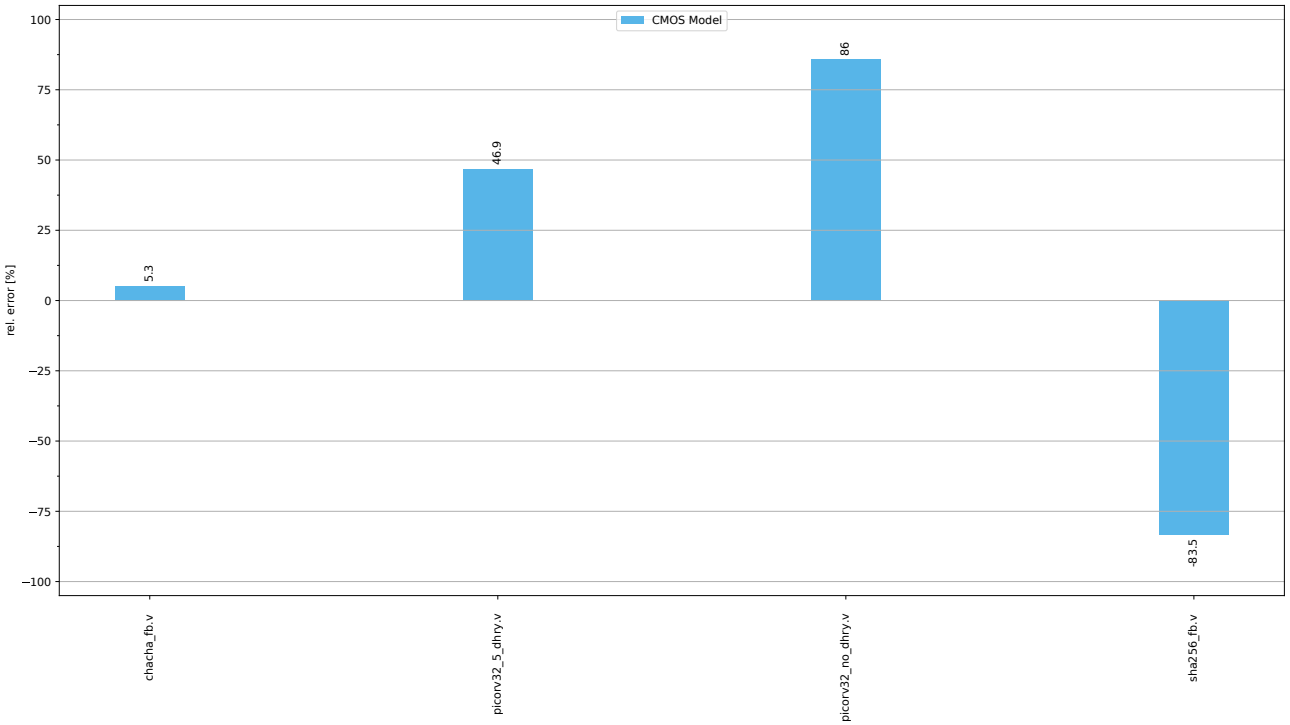


Figure 4.5: Relative error of the practical benchmarks estimated (CMOS model) power compared to the measured results.

more properties with our model. They are added to the plain **SB_DFF** stats as a simple measure to include them.

SB_DFFE	SB_DFFSR	SB_DFFR	SB_DFFSS	SB_DFFESR	SB_DFFER	SB_DFFESS	SB_DFFES
SB_DFFNE	SB_DFFNSR	SB_DFFNR	SB_DFFNSS	SB_DFFNESR	SB_DFFNER	SB_DFFNESS	SB_DFFNES

Table 4.1: Previously missing DFF variants.

4.2.4 Component matrix analysis

After all these fixes we gaze at the component matrix $\boldsymbol{A}$ Table 4.2 for our practical benchmarks, which gives us some insights into the behavior of our benchmarks. Taking a look at $\boldsymbol{A}_{\text{ChaCha}}$ one can see that some activation rates are zero. This is very abnormal and upon taking a look into the simulation trace we see that the **ChaCha** core did not work properly. The defective behavior only occurred after synthesis, which shows that simulating a synthesized design is a very important to ensure proper functionality on hardware. This means the results from this core are ignored and retaken for proper comparison.

	$U^2 f_{\text{LUT,I0}}$	$U^2 f_{\text{LUT,I1}}$	$U^2 f_{\text{LUT,I2}}$	$U^2 f_{\text{LUT,I3}}$	$U^2 f_{\text{LUT,O}}$	$U^2 f_{\text{DFF,C}}$	$U^2 f_{\text{DFF,D}}$	$U^2 f_{\text{DFF,Q}}$	$U^2 f_{\text{CARRY,CI}}$	$U^2 f_{\text{CARRY,I0}}$	$U^2 f_{\text{CARRY,I1}}$	$U^2 f_{\text{CARRY,CO}}$	Us_{LUT}	Us_{DFF}	Us_{CARRY}
$\boldsymbol{A}_{\text{ChaCha}}$	0	0	0	2.18e+08	3.27e+08	3.51e+11	1.09e+08	1.09e+08	0	0	0	0	1.23e+04	5.32e+03	3.46e+03
$\boldsymbol{A}_{\text{Ice,idle}}$	7.83e+10	1.66e+11	1.97e+11	2.24e+11	2.21e+11	2.34e+12	4.29e+10	3.33e+10	2.48e+10	1.02e+10	1.35e+10	2.26e+10	1.5e+04	3.55e+03	3.62e+03
$\boldsymbol{A}_{\text{Ice,Dhry}}$	6.64e+10	1.46e+11	1.76e+11	2.24e+11	1.93e+11	2.34e+12	3.64e+10	2.75e+10	2.14e+10	9.04e+09	7.88e+09	1.94e+10	1.5e+04	3.55e+03	3.62e+03
$\boldsymbol{A}_{\text{SHA256}}$	2.5e+10	1.13e+11	9.52e+10	1.06e+11	1.64e+11	2.81e+11	4.79e+10	3.34e+10	3.33e+10	2.48e+10	2.28e+10	3.43e+10	1.01e+04	4.26e+03	1.25e+03

Table 4.2: Component activation and utilization matrices from all 4 practical benchmarks.

4.3 Power Estimation of use cases

For proper validation of the qualitative performance of our estimation we pick designs of different complex applications. The biggest constraint for the design selection is that it has to fit onto the Lattice iCE40up5k, which a few designs that we want to consider violate. The final selection of practical designs for the real world accuracy and usability analysis consist of a RISC-V CPU, a stream cipher and a cryptographic hash algorithm.

4.3.1 Use case 1 – PicoRV32

This CPU core is designed for a small size footprint, which makes it ideal to use on our low LUT count FPGA. For this evaluation of our estimator we simply use the provided **picosoc** implementation, which adds some additional periphery to allow communication to the system via Universal Asynchronous Receiver/Transmitter (UART). The software running on the core is the also included Dhrystone benchmark.

4.3.2 Use case 2 – SHA256

For a hash we want to select the more modern BLAKE3 algorithm. Unfortunately this design did not fit into our LUT budget. Hence we use the very common **SHA256**. The testbench setup is based on a self looping design as shown in Figure 4.7. This means the algorithm is fed an initial state and then mutates that input by hashing it and uses that output as its new input.

4.3.3 Use case 3 – ChaCha

Initially we look into an **AES** core, but this does not fit onto the FPGA. After further search we settle with the popular symmetric **ChaCha** stream cipher. The cipher core is implemented in a similar self looping fashion as the hash design with a fixed key. Due to the cipher being symmetrical this results in continuous encrypting and decrypting the initial message. This is neatly observed when simulating the testbench.

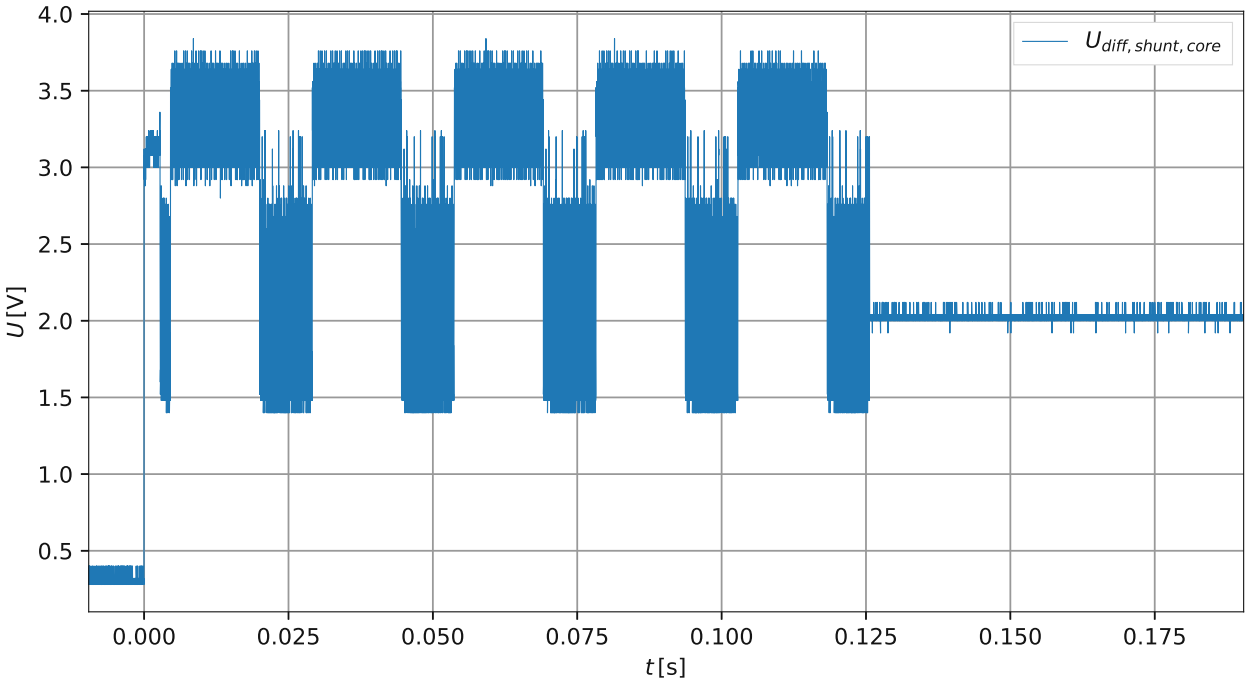


Figure 4.6: Measurement of Picorv32 running the Dhrystone benchmark 5 times

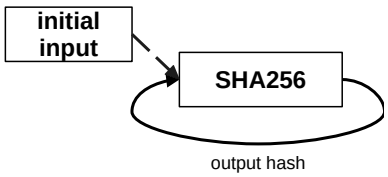


Figure 4.7: Testbench of the self looping SHA256 benchmark

4.3.4 Vendor Tool Measurements

For a proper evaluation of our method we compare our results with an estimator provided by the vendor tooling. Due to differences in synthesis the results do not translate exactly, but with similar resource consumption they should return a comparable value. Using the practical benchmarks from before we synthesize them with the vendor tool. The LUT usage and power consumption for each design is shown in Table 4.3. The **ChaCha** and **SHA256** benchmarks both have a similar hardware utilization as when synthesized with Yosys, which is a good indicator for comparability to our results. The Picorv32 design uses some architecture specific primitives, which the vendor tool seems to support with some workarounds, but the synthesis resulted in high consumption of LUTs. We are unable to fix this benchmark to allow the mapping onto the FPGA. This makes the benchmark not usable for the comparison.

	Picorv32	ChaCha	SHA256
SB_LUT4	52435	3734	3044
Static Power [mW]	0.087	0.519	0.442
Dynamic Power [mW]	-	6.383	5.55
Total Power [mW]	-	6.90	6

Table 4.3: Hardware utilization and power estimation of the practical benchmarks with vendor tool.

4.3.5 Comparison

Table 4.4 provides a comparison between the vendor tool and our estimates. It also provides the measurements from the designs synthesized by Yosys. This means the measurements should be considered the ground truth to our estimates. For the vendor tool they don't correctly represent the hardware power consumption from the synthesized designs, since we don't know what exactly their estimates include. The estimates of **ChaCha** and **SHA256** on both tools resulted in power of the same order of magnitude, but they are a few factors off from the hardware measurements. The **SHA256** is the design with the largest discrepancy. It has a relative error of 75 % for our and 90 % for the vendor estimate. Initially we thought that this might be a measurement error. We check the synthesis again and remeasured the power on hardware, but we still get a similar value. The **ChaCha** design's estimates on the other hand seem less like outliers with 143 % of relative error for our solution and 109 % for the vendor's.

	Vendor			Ours		
	Picorv32	ChaCha	SHA256	Picorv32	ChaCha	SHA256
SB_LUT4	52435	3734	3044	4282	3728	3066
Total Power [mW]	-	6.9	6	31.6	5.6	9.5
Measured Power [mW]	-	3.3	64	11.8	2.3	38.4
Relative error [%]	-	109	90	167	143	75

Table 4.4: Hardware utilization and power estimation of the practical benchmarks with vendor tool and our solution.

Chapter 5

Conclusion

Our proposed methodology provides a flow to characterize a given FPGA architecture and in the end to estimate the power of new hardware designs. With test benchmarks we offer the basis to characterize parts of a FPGA hardware that they are synthesized onto. The testbenches provide the needed activation rate data and describe the requirements for the top level design for running the benchmarks on hardware. With our measurement work flow in combination with testbenches and characterization benchmarks we are able to collect power measurements that are needed for fitting . We fit this data utilizing a CMOS model using our design analyzer tools.

For direct use with our results and data, one has to some constraints. A Lattice iCE40up5k based FPGA has to be used and one has to provide a simulation testbench for the design that is estimated. With the use case benchmarks we show that our method works quite well, given the limited fitting data sets and low complexity of our approach. In comparison to the vendor tool, it works on a similar level of accuracy with the relative error of our worst estimate only differing by 34 %. Our best result even surpasses the vendor estimate by 15 % in relative error.

This work can provide a basis for further work on the topic of FPGA power estimation by highlighting certain pitfalls and providing useful ideas on how to approach some aspects.

Bibliography

[1] J.H. Anderson and F.N. Najm. "Power estimation techniques for FPGAs". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12.10 (Oct. 2004), pp. 1015–1027.

[2] David Elléouet, Yannig Savary, and Nathalie Julien. "An FPGA Power Aware Design Flow". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 415–424.

[3] Ruzica Jevtic and Carlos Carreras. "Power Measurement Methodology for FPGA Devices". In: *IEEE Transactions on Instrumentation and Measurement* 60.1 (Jan. 2011), pp. 237–247.

[4] Hyung Gyu Lee, Kyungsoo Lee, Yongseok Choi, and Naehyuck Chang. "Cycle-Accurate Energy Measurement and Characterization of FPGAs". In: *Analog Integrated Circuits and Signal Processing* 42.3 (Mar. 2005), pp. 239–251.

[5] C. Najoua, B. Mohamed, and B.M. Hedi. "Power estimation model based on grouping components in field-programmable gate array circuit". In: *IET Circuits, Devices & Systems* 6.6 (Nov. 2012), pp. 437–446.

[6] Stefan Riesenberger and Christian Krieg. "Towards Power Characterization of FPGA Architectures To Enable Open-Source Power Estimation Using Micro-Benchmarks". In: Proceedings of the 3rd Workshop on Open-Source Design Automation (OSDA), 2023, co-hosted with Design, Automation, and Test in Europe (DATE) conference in Antwerp, Belgium, April 17, 2023.

[7] Mohammad Y Al-Shorman, Majd M Al-Kofahi, and Osameh M Al-Kofahi. "A practical microwatt-meter for electrical energy measurement in programmable devices". In: *Measurement and Control* 51.9-10 (Aug. 2018), pp. 383–395.

[8] Gaurav Verma, Vijay Khare, and Manish Kumar. "More Precise FPGA Power Estimation and Validation Tool (FPEV_Tool) for Low Power Applications". In: *Wireless Personal Communications* 106.4 (Sept. 2018), pp. 2237–2246.

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, Austria March 13, 2025

Stefan Riesenberger