NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENT OF

ELECTRICAL AND COMPUTER ENGINEERING

TIAGO JORGE GUERREIRO DA SILVA

BSc in Electrical and Computer Engineering

Implementation study of a RISC-V based SoC for IoT using SKY130 open PDK

DISSERTATION TO OBTAIN THE DEGREE OF MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

September, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENT OF

ELECTRICAL AND COMPUTER ENGINEERING

Implementation study of a RISC-V based SoC for IoT using SKY130 open PDK

DISSERTATION TO OBTAIN THE DEGREE OF MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

TIAGO JORGE GUERREIRO DA SILVA

BSc in Electrical and Computer Engineering

Adviser: Professor João Pedro Oliveira
Professor, NOVA University Lisbon

Examination Committee:

Chair:	Doutor Filipe de Carvalho Moutinho - FCT/UNL
Rapporteurs:	Doutor Rui Manuel Leitão Santos Tavares - FCT/UNL
Adviser:	Doutor João Pedro Abreu de Oliveira - FCT/UNL

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
September, 2022

Implementation study of a RISC-V based SoC for IoT using SKY130 open PDK.

ACKNOWLEDGEMENTS

I would like to give special thanks to all the professors that accompanied me throughout my whole school life, especially those whose eyes sparkled when sharing their knowledge with me. To my adviser Professor João Pedro Oliveira, who challenged me into such modern and unexplored topic, and whose passion for teaching could be felt since the very first class.

To the people at The Inventors who reignited my passion for engineering, and to my colleagues and students at Sharkcoders who remind me of my love for teaching.

To all the ones I ever called friends, whose presence shaped me into who I am today. I would like to particularly thank Tiago Vicente, for being the very best and oldest of friends, and Diogo Maymone, as were it not for you I wouldn't have gone on the trip of my life.

To my family, particularly my mother for being my superhero, my father for teaching me so much, and my sister for all the nightlong talks about nothing and everything. Thank you all for always, always, caring and being there to support me in my times of need, even if you don't quite understand what a transistor is.

And finally, to Patricia, for being my strength and covering my weaknesses. I will not forget the countless days you helped organize and improve this document, and the sheer amount of synonyms you taught me during this journey. But, still, you could, you know, learn to, use less comas.

“An earnest failure has meaning”

ABSTRACT

Hardware design has always been an area with a tall barrier of entry. Aside from requiring high expertise, it is plagued with proprietary tools and IPs. However, the open-source community has been striving to change this for over a decade now, by developing alternatives and publishing them on open licenses. A key achievement was the arrival of RISC-V, that offered the possibility of developing processors with no fees. Nowadays, this movement is reaching chip design through free IPs and recently open access to Process Design Kits (PDK) and Electronic Design Automation (EDA) tools.

This dissertation contributes to open-source by proposing a design flow that enables System on Chip (SoC) development using RISC-V. This flow is comprised of freely accessible tools and is aimed at students and researchers. For the logic design it uses Vivado, Visual Studio Code with PlatformIO, Verilator, and GTKWave, along with the SwervolfX SoC that can be implemented into a FPGA. This provides an environment to test and develop peripherals for a RISC-V system. The physical design uses Openlane, which is compatible with Skywater's open 130nm PDK.

To test the flow, a biquad filter was hardened into a 0.185 mm^2 macro, then integrated into the Caravel SoC and submitted to Google's open MPW shuttle.

Keywords: RISC-V, Open PDK, Openlane, OpenROAD, Open-source, SoC, IoT, FPGA, EDA, Digital Design Flow, ASIC, DSP, Biquad filter, Open MPW-7, Caravel.

RESUMO

O desenho de hardware foi sempre uma área com uma barreira de entrada alta. Para além de exigir elevada perícia, está infestada de ferramentas proprietárias e IPs. No entanto, a comunidade de código aberto tem procurado alterar esta situação há mais de uma década, desenvolvendo alternativas e publicando-as em licenças abertas. Uma conquista chave foi a chegada do RISC-V, que ofereceu a possibilidade de desenvolver processadores sem taxas. Atualmente, este movimento está a chegar ao desenho de chips através de IPs gratuitos e, recentemente, o acesso aberto a "Process Design Kits" (PDK) e ferramentas de "Electronic Design Automation" (EDA).

Esta dissertação contribui para o código aberto ao propor um fluxo de desenho que permite o desenvolvimento de Sistema em Chip (SoC) utilizando RISC-V. Este fluxo é composto por ferramentas de livre acesso e destina-se a estudantes e investigadores. Para o desenho lógico, utiliza Vivado, Visual Studio Code com PlatformIO, Verilator, e GTKWave, juntamente com o SoC SwervolfX que pode ser implementado numa FPGA. Isto proporciona um ambiente para testar e desenvolver periféricos para um sistema RISC-V. O desenho físico utiliza o Openlane, que é compatível com o PDK aberto de 130nm da Skywater.

Para testar o fluxo, um filtro biquad foi endurecido numa macro de 0,185 mm2 e depois integrado no SoC "Caravel" e submetido ao programa MPW aberto do Google.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND INITIALISMS

ADC	Analog-To-Digital Converter
AFE	Analog Front-End
ASIC	Application Specific Integrated Circuits
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CISC	Complex Instruction Set Computer
CMOS	Complementary metal–oxide–semiconductor
CPU	Central Processing Unit
CTS	Clock Tree Synthesis
CVC	Circuit Validity Checks
DFT	Designs for Testability
DRC	Design Rule Check
ECG	Electrocardiogram
EDA	Electronic Design Automation
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HV	High Voltage
IC	Integrated Circuit
IIR	Infinite-Impulse-Response
IO	Input Output
IoT	Internet of Things

ISA	Instruction Set Architecture
LCD	Liquid Crystal Display
LVS	Layout vs Schematic
MCM	Multiple Chip Module
MiM	Metal-Insulator-Metal
MIPS	Microprocessor without Interlocked Pipelined Stages
MPW	Multi Project Wafer
NMOS	N-channel metal-oxide semiconductor
PDK	Process Design Kit
PDN	Power Delivery Network
PMOS	P-channel metal-oxide semiconductor
POWER	Performance Optimization With Enhanced RISC
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SoB	System on Board
SoC	System-on-Chip
SONOS	Silicon–Oxide–Nitride–Oxide–Silicon
SPARC	Scalable Processor Architecture
STA	Static Timing Analysis
VHDL	Very High-Speed Integrated Circuits Hardware Description Language
WCSP	Wafer Chip Scale Package

1 INTRODUCTION

This chapter begins by providing the motivations and objectives of this work. Afterwards it explains the main contributions. Lastly, the organization of the thesis is presented.

1.1 Motivation and Background

In todays' age almost every electronic device has a processor or controller in it. Some, like computers and automobiles even have dozens of them to perform specific tasks. This makes integrated circuit design, especially application specific ones (ASIC), more important than ever before.

However, the market is completely saturated with proprietary chips, mostly from Intel and ARM, and it's extremely costly for anyone outside of those companies to design their own chips, due to licencing fees. That is where RISC-V comes in. Above all else, RISC-V's true value lies in its open-source nature, meaning that anyone, anywhere, can design and sell a chip based around its architecture completely free of charge.

The popularization of open-source software began after the appearance of Linux revolutionized the software ecosystem. This meant anyone with an internet connection could actively participate in the development of software. RISC-V is showing signs of doing the same but for the hardware community.

There have been countless companies and organizations joining the RISC-V movement and contributing to the open-source hardware ecosystem. A major milestone for this was the partnership between Google and Skywater Technology in June 2002, which resulted in release of the industry's first open-source process design kit (PDK), known as SKY130 (130nm CMOS technology).

Another important element of ASIC design are the EDA (Electronic Design Automation) tools. The open-source options have also evolved over the years, and today there are several environments to choose from. The recent addition, Openlane, is focusing around the beforementioned sky130 PDK, which together with RISC-V provides a full open-source alternative do ASIC design.

Although the previous alternative now exists, the manufacturing of chips is still very expensive and a tall barrier of entry to ASIC enthusiasts. Luckily, Google's partnership with Skywater Technology goes further than this. With the help of Efabless[1], the open MPW (Multi Project Wafer) program was

[1] Efabless is a company that specializes in accelerating the chip fabricating process of its users.

created, where people can submit their ASIC designs to be manufactured at no cost, provided they are licensed as open-source.

1.2 Objectives and Contributions

The main objective of this project is to make an assessment on the feasibility of developing an in-house system-on-chip (SoC) using only the available open-source tools and opportunities. If deemed feasible, an original design flow will be proposed.

Another objective is to test and synthesize a digital biquadratic filter and finally integrate it into a SoC provided by Efabless. The resulting SoC was submitted to the open MPW program to be manufactured.

One last objective is to provide a practical tutorial on the used open-source tools, mainly Openlane.

1.3 Thesis Organization

This document is organized as follows.

Chapter 2 offers an introduction to the concepts needed for understanding this work and presents the state of the art whenever relevant. The topics covered are System on Chips (SoC), the generic Applications Specific Integrated Circuit (ASIC) design flow, and RISC-V.

In Chapter 3 there is presented an overview of the open-source tools scene. It goes into greater detail about the Openlane flow, providing a user guide to the environment and main functionalities. The SKY130 PDK is also characterized there. Lastly, the open MPW shuttle program is explained.

Chapter 4 is where the proposed open design flow is given. Later in the same chapter the results of hardening a biquad filter with said flow are provided. The process of integrating macros into SoCs is also exemplified there.

In Chapter 5 the conclusions of the work are drawn, and future work is proposed.

2 BACKGROUND ON CHIPS

This chapter serves as an introduction to the topics covered by this project.

Figure 2-1 represents a diagram of its structure.

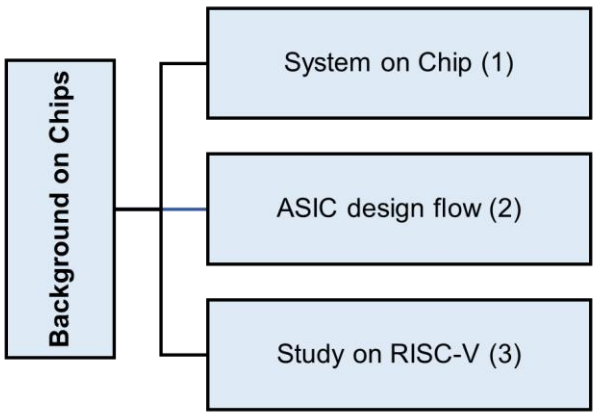


Figure 2-1 – Background on chips chapter structure.

The first subchapter presents an introduction into System on Chips (SoC) as well as a small study on modern day ones.

Following it is a section that provides an overview of Application Specific Integrated Circuits (ASIC) design flow and explains its various steps.

Lastly, a study and explanation on RISC-V will be given, showing current implementations, and comparing it to other Instruction Set Architectures (ISA).

2.1 System on Chip

An electronic system is a pretty self-explanatory term. It is a system that is composed entirely of electronic components. These can be used for many applications and integrated into other non-electronic systems, so they are present into almost every existing industry. As a result of this, peoples' everyday lives are surrounded by electronics.

These systems are always composed by multiple elements, each with its own task. Take a desktop computer for example. It's composed of a CPU, RAM, Storage Device, and a Motherboard, between others. All of these modules come together to create to create a system known as a computer.

A system on chip (SoC) is just like that, except all its elements reside inside a single integrated circuit (IC). SoC have existed since as early as the 1970s and have been evolving ever since [1]. Today, they're used in a wide range of applications such as embedded systems and mobile and personal computers. Figure 2-2 shows the first commercial SoC.

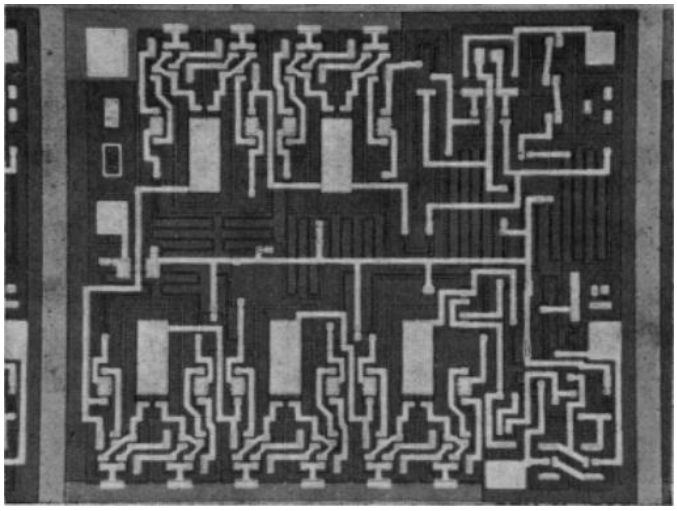


Figure 2-2 – Photomicrograph of the first commercial watch SoC. Extracted from [1].

As its obvious electronics are an ever so important part of the current world, there is a growing demand for higher performance electronics systems. One way engineers are achieving this is by adopting the use of SoCs. To realize why the industry is shifting to this method it's important to understand its advantages and disadvantages when compared to a classic multiple chip module (MCM) or system on board (SoB) approach.

The true purpose of a SoC is that it enables optimization in a way that was previously impossible. By confining the whole system inside an IC, it effectively shrinks it to the size of the chip, reducing manufacturing costs and allowing it to be used in applications where small size is mandatory. Such applications generally have reduced space for batteries as well, so SoCs are usually very optimized for power consumption. Another important advantage of the smaller size is that now all components are closer together, meaning that, with proper planning, latency can be greatly reduced improving performance. However, smaller size also means higher heat density, which can potentially bottleneck or even damage the IC. This flaw emphasizes the aforementioned need for low energy consumption, as less power means less heat generation. SoC are also harder or downright impossible to repair/upgrade a faulty or old module in its system.

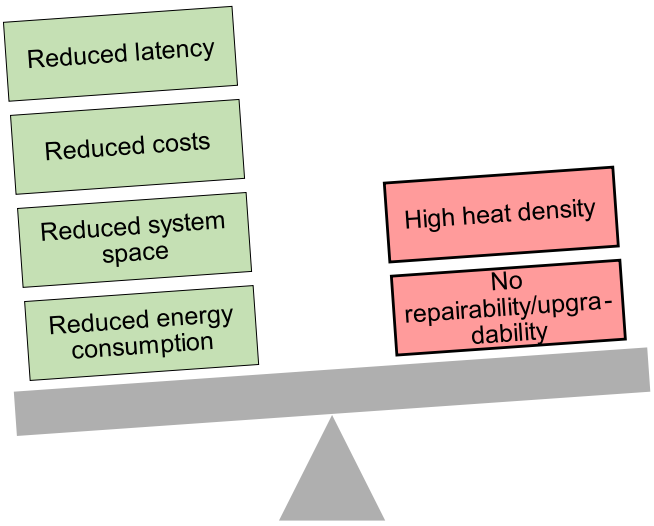


Figure 2-3 – SoC Analysis.

As easily observed in Figure 2-3,despite SoC's having a few shortcomings, these are far outweighed by their advantages, supporting the industry's growing tendency to this method.

SoC Architecture

There is an infinite amount of SoCs that vary depending on the system's specific application. It's worth noting that the technological advancements allowed the integration of many new components to the chip architecture. As such, modern SoCs are vastly different now than they were 50 years ago, and each chip can nowadays contain a higher number of modules. A few examples will be presented later in this section.

Although it's hard to break down exactly what constitutes a SoC, almost all components are part of either one of the following two groups:

- CPU subsystem: composed by the CPU and blocks designed to aid or accelerate it. It's tasked with assuring the correct behavior of the system and capable of running software. All composing elements communicate via a system interconnect.
- Peripherals: whether digital, analog, or mixed signal, these are all the blocks with specific functions that reside outside of the CPU subsystem and usually connect to it using a special peripheral interconnect.

Out of all the existing possible modules that constitute a modern SoC, there are four that are present in almost all of them, no matter the application. These can be observed in Figure 2-4.

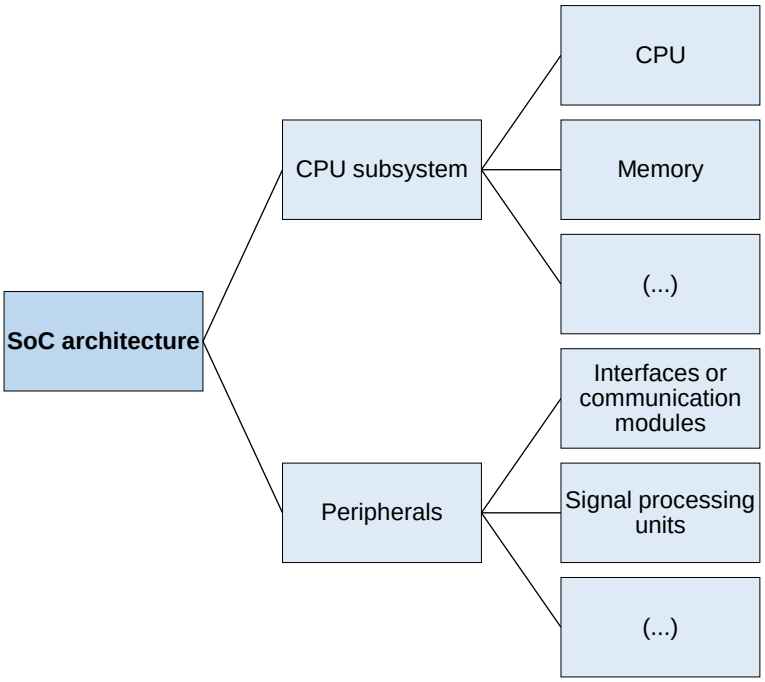


Figure 2-4 – SoC Architecture Elements.

IoT applications – PSoC 64

An example of a modern SoC is the PSoC 64, the latest model of the PSoC series. This chip was developed by Cypress[2], an American semiconductor design and manufacturing company [2]. The PSoC 6 family was purpose-built for the IoT, specifically battery powered applications. The 64 version was particularly designed for those that require high security [3].

Figure 2-5 shows its architecture, where the CPU subsystem is highlighted in green and the peripherals in red.

[2] Cypress was acquired by Infineon Technologies in 2020.

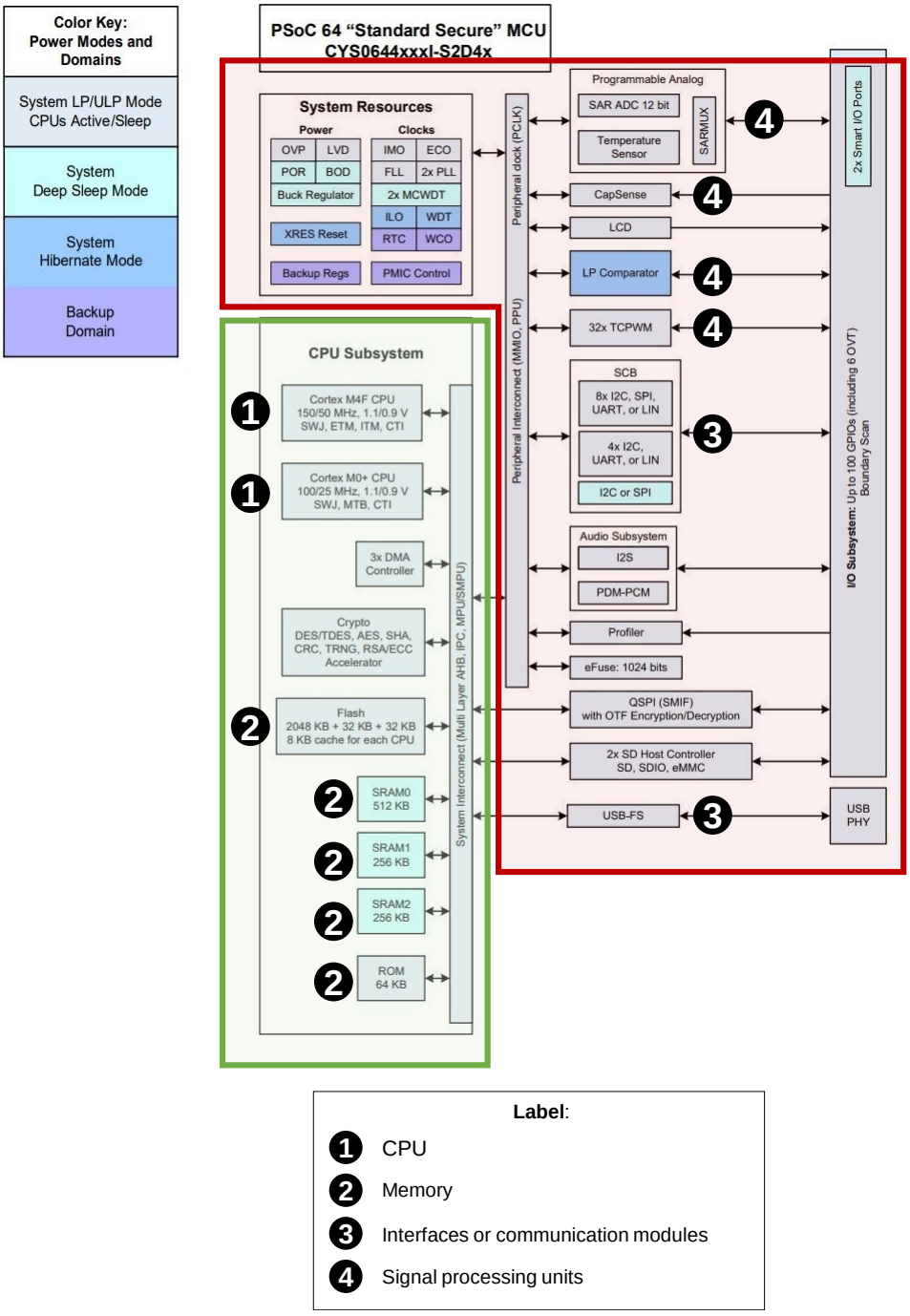
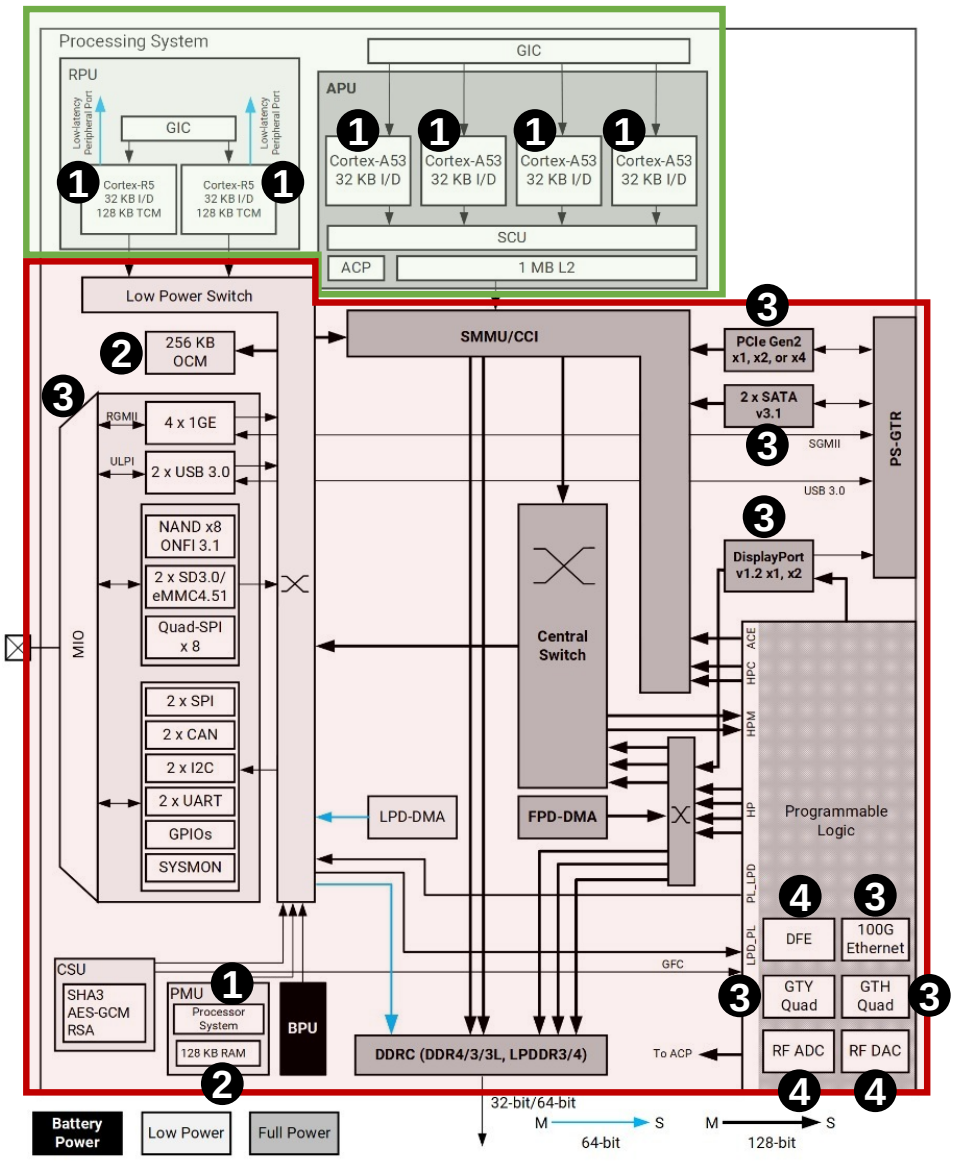


Figure 2-5 – PSoC 64 Architecture.

Radio frequency applications – Zynq UltraScale+ RFSoC

The Zynq UltraScale+ RFSoC is another modern SoC. Created by Xilinx AMD, it is used essentially for testing and building radio frequency applications. A distinguishing factor is that this chip integrates programable logic in the form of a FPGA which increases design flexibility [4].

Following the previous example, Figure 2-6 shows RFSoC's architecture.



Label:

1. CPU
2. Memory
3. Interfaces or communication modules
4. Signal processing units

Figure 2-6 – Zynq Ultrascale+ RFSoC Architecture.

2.2 ASIC design flow

Since this project seeks to create a SoC, this subchapter will serve as an introduction to chip design, breaking down the process and shortly describing its various steps. This process is also known as ASIC (Application Specific Integrated Circuit) design flow and usually takes several teams working together many months to complete. It is normally divided into two major stages:

- Logic design: commonly described as the frontend.
- Physical design: commonly described as the backend.

Figure 2-7 provides an overview of the whole flow.

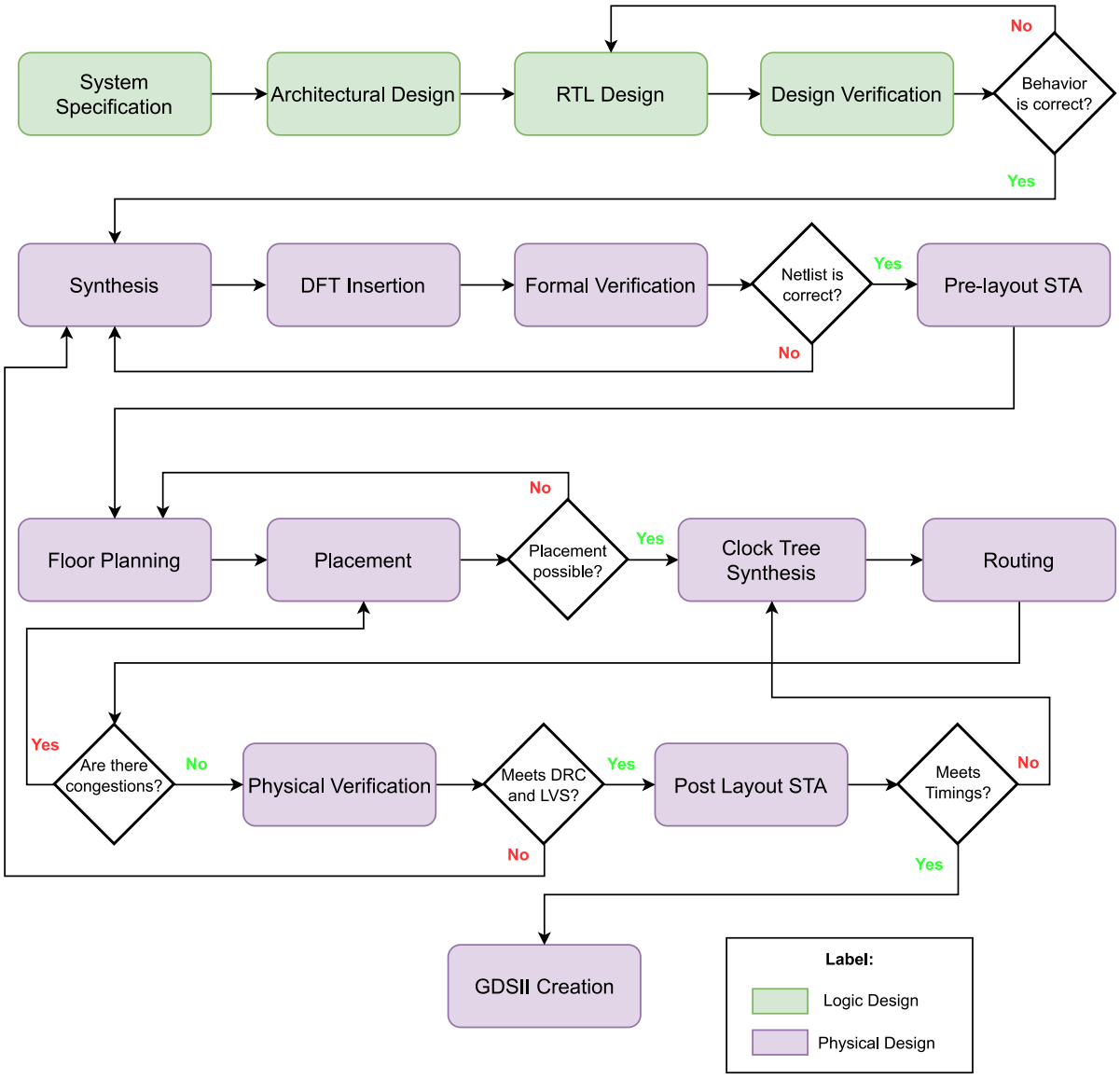


Figure 2-7 – ASIC design workflow.

2.2.1 Logic Design

The Logic Design is the first main group of steps that constitute the ASIC design flow. It involves all the steps that face the chip at a more logic/behavioral level as opposed to the physical level. As mentioned before, it includes the following four stages, shown in Figure 2-8, that will be discussed one by one.

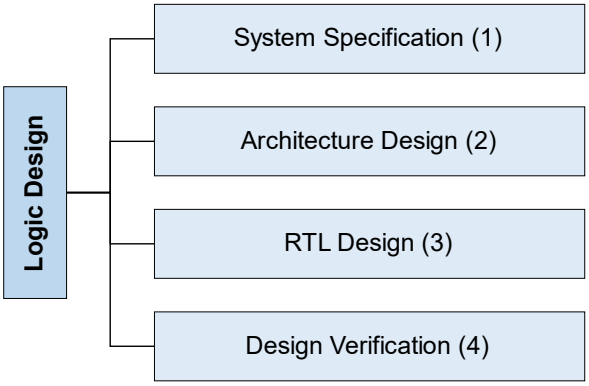


Figure 2-8 – Logic design steps.

System Specification (1)

The first step in any IC's design flow is to define its specifications. ASIC design and manufacturing cycle are lengthy processes. Therefore, developing a product that will still be relevant even after several years is of utmost importance, making it crucial to conduct extensive market research at this point.

This then translates into high-level product specification such as functionality, chip dimensions, package type, power supply, clock frequencies, and IO pins, between others.

The operational conditions should also be set here, as these have an impact in the fabrication process. The following Figure 2-9, that is part of the 74HC157 Multiplexer datasheet, shows an example of some such conditions that should be set in this stage.

Operating Conditions

Temperature Range (T_A)	-55°C to 125°C
Supply Voltage Range, V_{CC}	
HC Types	2V to 6V
HCT Types	4.5V to 5.5V
DC Input or Output Voltage, V_I, V_O	0V to V_{CC}
Input Rise and Fall Time	
2V	1000ns (Max)
4.5V	500ns (Max)
6V	400ns (Max)

Figure 2-9 – 74HC157 Operational Conditions.

Architecture Design (2)

After setting the specifications, the ASIC is divided by functionality into multiple blocks. There are usually many different possible partition arrangements that result in different architectures, but that doesn't mean this is an easy job. On the contrary, to achieve a good architecture one must balance performance, technical feasibility, and chip size, while also keeping costs down (both time and money). This applies to both full chip architectures, and smaller, single design architectures. Figure 2-10 and Figure 2-11 show different architectures of a multi-cycle 4-bit multiplier module, as an example.

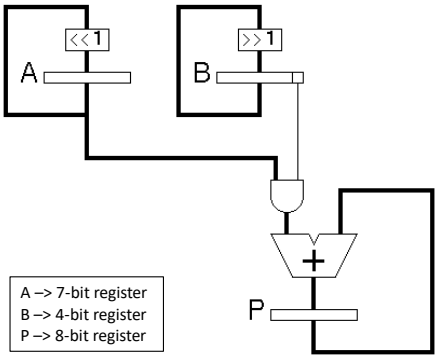


Figure 2-10 – Multiplier Architecture 1.

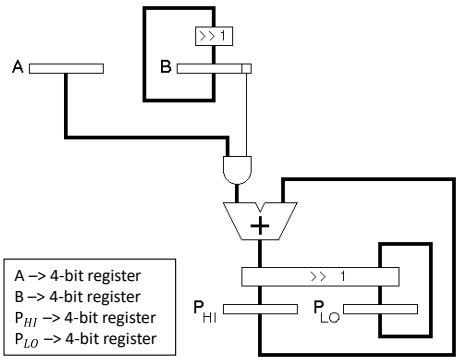


Figure 2-11 – Multiplier Architecture 2.

Examples of complete SoC architectures can be found in the previous Figure 2-5 and Figure 2-6 in chapter 2.

RTL Design (3)

Register Transfer Level (RTL) design is when the system functionalities are broken down into Boolean equations and then written down as HDL (Hardware Description Language), such as VHDL, Verilog, or SystemVerilog. These languages not only make digital design easier for less experience engineers, but also greatly speed up the process. It also enables specialized programs to quickly convert the system into physical logic gates. While not calculating delays at this point, they should still be considered when writing the RTL code.

```
module mux (a, b, c, d, s0, s1, out);
        input wire a, b, c, d;
        input wire s0, s1;
        output reg out;

        always @ (a or b or c or d or s0, s1)
                begin
                case ({s1,s0})
                        2'b00 : out <= a;
                        2'b01 : out <= b;
                        2'b10 : out <= c;
                        2'b11 : out <= d;
                endcase
```

```
        end
endmodule
```

Design Verification (4)

Design verification, also called functional verification, is where the RTL code previously written is tested, usually with the help of test vectors or testbenches. These must be able to test the design in its entirety to ensure correct functionality. Since chip manufacturing is a long and expensive process, it's important to ensure the design works exactly as intended and has no bugs. Therefore, this step is often done in parallel with RTL design and is one of the most important steps in the logic design flow. Although such an emphasis is put in verification, it's still very common for chips to have problems after fabrication. As such, other test methods like FPGA implementation, which tests a design in hardware rather than by simulations, are ever so important. An example of a testbench is offered in Figure 2-12.

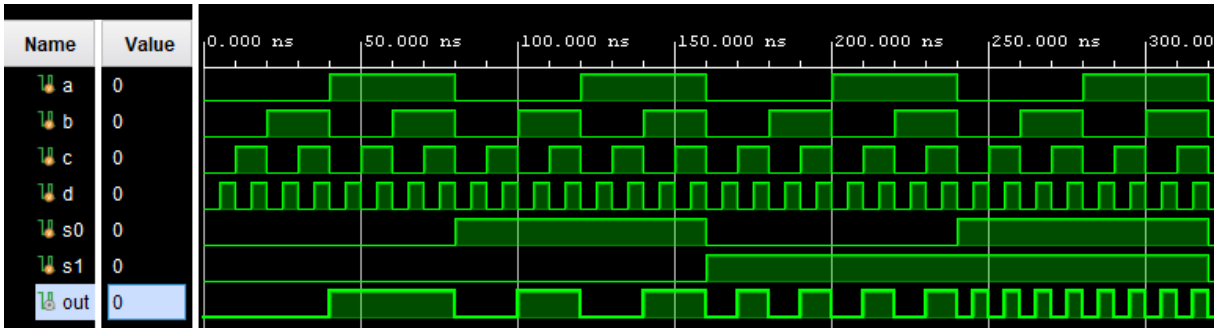


Figure 2-12 – Four input multiplexer testbench waveform.

2.2.2 Physical Design

Often also denominated as the backend cycle, this is when the physical aspects of the chip really start getting calculated and optimized. What distinguishes this the most from the previous frontend is that now delays are fully considered right from the beginning.

This cycle can be further divided into the following structure in Figure 2-13:

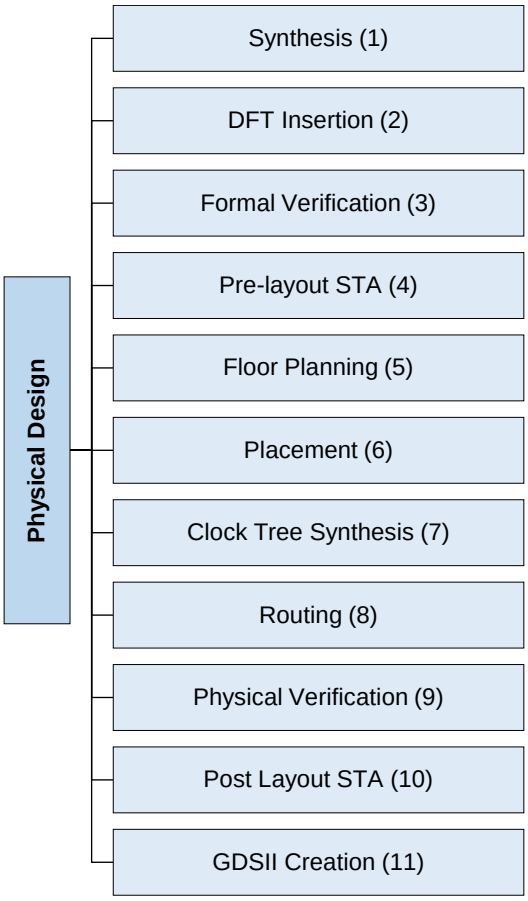


Figure 2-13 – Physical design steps.

Synthesis (1)

RTL code is translated into a gate-level netlist by a synthesis tool and converted into physical components like wires, gates, registers, and multiplexers. As there are often many possible implementations of a design, it's the tool's job to consider each case's power consumption, performance and area usage and choose the best based on previously defined priorities. In case the target constraints are still not met, both the architecture and the RTL design can be tweaked, using methods like resource sharing, pipelining, and dead zone elimination. Figure 2-14 shows an example of a 4-bit shift register netlist.

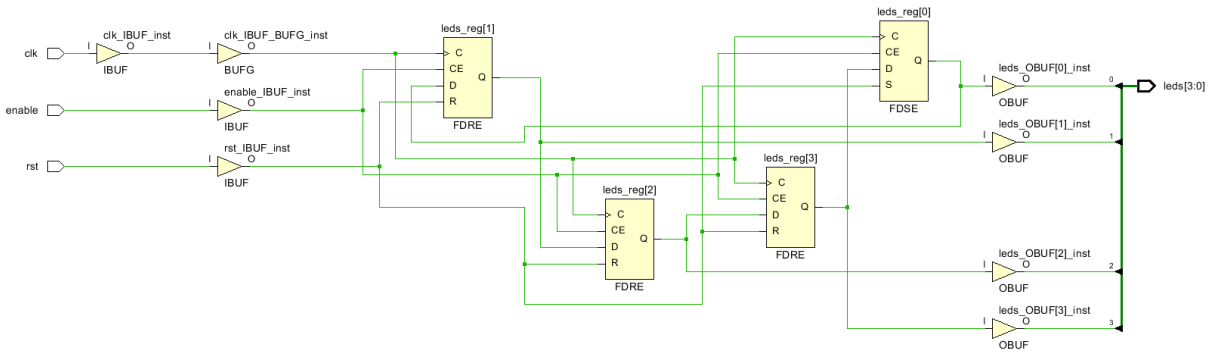


Figure 2-14 – 4-bit shift register netlist.

DFT Insertion (2)

Special circuits called Designs for Testability (DFT) are inserted into the netlist to help detect problems in the IC after it has been fabricated. Although this is not strictly necessary, it's good practice to do so, especially on bigger designs. Some techniques often used are:

- Memory BIST (Built-In Self-Test): widely used to check RAMs.
- Scan Path Insertion: the methodology of linking all registers into one long shift register. This helps check small areas of the IC one at a time, rather than the whole thing in one go.
- ATPG (Automatic Test Pattern Generation): a method of creating test vectors to help check for design faults.

Formal Verification (3)

The netlist generated in the synthesis phase needs to be verified to see if it implements the RTL code correctly. To do this, both the netlist and the RTL code are reduced to their Boolean equations. If they are the same, then the netlist is correct.

Pre-layout STA (4)

The objective of the STA or Static Timing Analysis is to identify potential timing violations on the design, mainly setup and hold. Setup and hold time are, respectively, the duration a signal must be stable before and after the clock's active edge to ensure correct the data is latched. During this pre-layout phase only setup violations are fixed since the design doesn't have routing information yet.

Floor Planning (5)

This process takes the floorplan that was made during the Architecture Design step and expands on it. The whole die area is divided into physical partitions, each one usually corresponding to a module of the system. The shapes and locations of these partitions need to be carefully considered as they greatly influence the next steps. A good floorplan is one that tries to minimize area, keeps congestions at a minimum, and reduces delays. Pin placement and power planning are also done in this phase. It's common practice to isolate digital circuitry from analog whenever possible, since the latter is usually very sensitive to noise and the former is a big source of it.

As this is probably the most important step in the physical design process, it's very common for engineers to work on more than one floorplan in parallel, and to do multiple iterations of them. Figure 2-15 provides a graphical example of a floorplan.

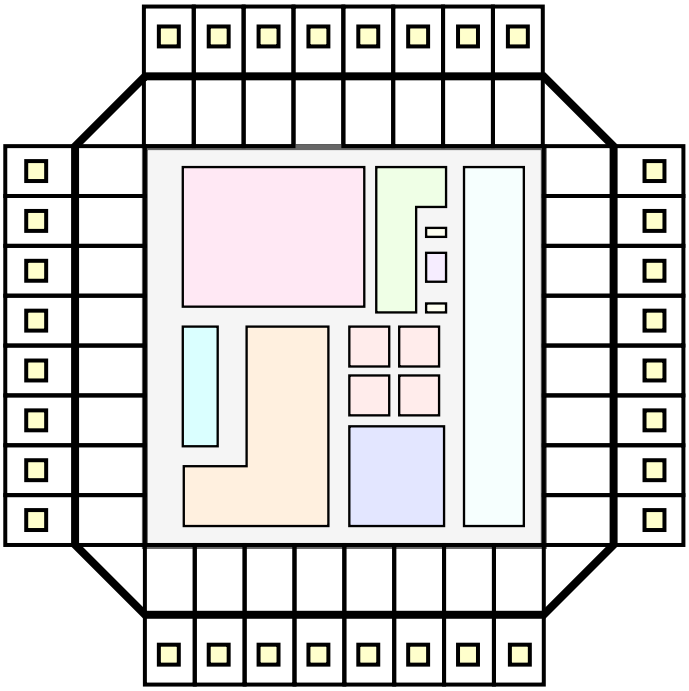


Figure 2-15 – Floorplaning.

Placement (6)

Standard cells are placed inside the partitions created in the previous phase, like observable in Figure 2-16. The goal of this step is to minimize wire length while leaving enough space for routing. It's a very critical step, as poor placement can degrade performance and require larger area.

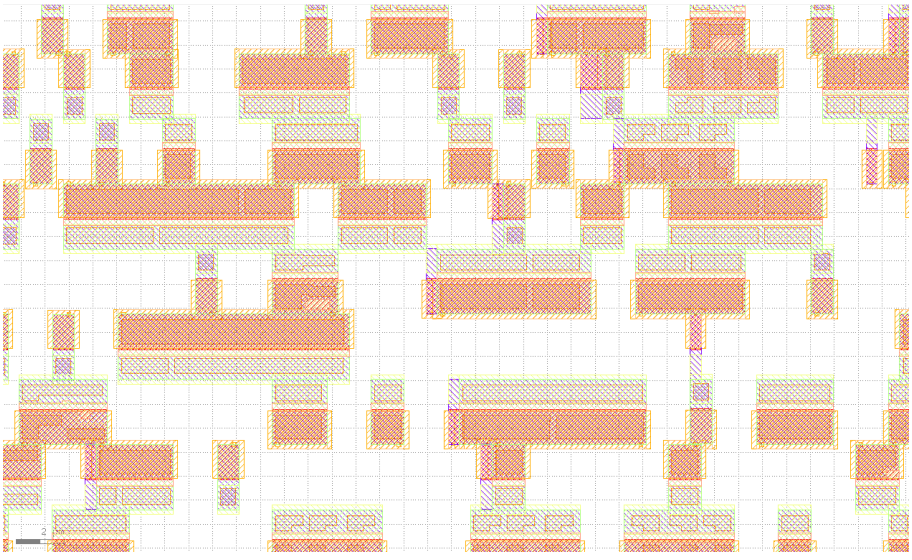


Figure 2-16 – Standard cell placement.

Clock Tree Synthesis (7)

The Clock is routed to different cells of the chip. It is done at a special, pre-defined layer to avoid delays on the nets of the IC. The main objective here is to achieve optimal clock latency while minimizing

clock skew. There are several structures that can be used to optimize power consumption and delays, such as Mesh, H-tree (Figure 2-17), X-Tree, Fishbone and Hybrid.

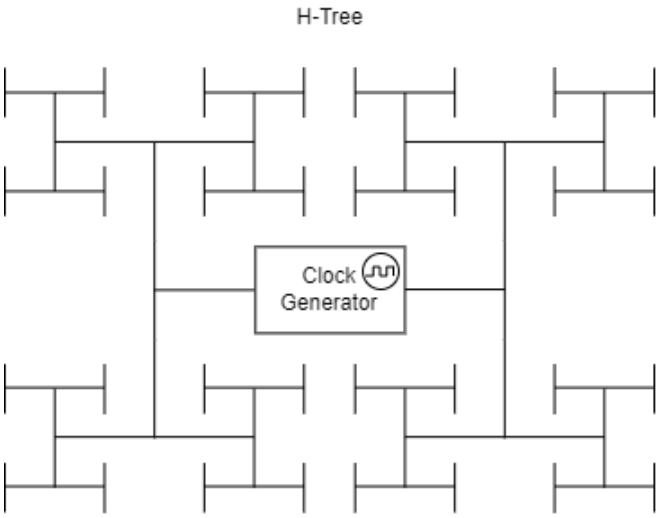


Figure 2-17 – H-Tree structure.

Routing (8)

Physical connections are created based on logical connectivity using metal interconnects. This step is usually done by a very specialized and efficient tool since it's a very computationally intensive job. The routed paths must meet requirements such as Timing, Clock skew, Maximum capacitance, and Physical Design Rule Check (DRC).

There are four phases in routing operations:

Table 2.1 – Routing stage phases.

Phase	Description
Global Routing	Nets are assigned to specific metal layers and routing cells. Global Route should avoid congested global cells while also minimizing detours.
Track Assignment	Each net is assigned to a specific track and metal traces are then laid down by it. Design rule check is not followed in this stage.
Detailed Routing	Actual delays of wires are calculated. It tries to fix all DRC violations using a small, fixed area known as a "SBox". The whole design is checked "box by box".
Search and Repair	This fixes all remaining DRC violations through multiple loops using progressively larger boxes.

Physical Verification (9)

The two most important checks, also known as sign-off checks, are done in this step:

- Layout vs Schematic (LVS): verifies whether the geometry/layout matches the schematic/netlist.

- Design Rule Check: verifies whether the design meets the rules given by the foundry.

Other verifications include Antenna and Electro-static Discharge violations, as well as shorts, opens and floating nets, between others.

Post Layout STA (10)

After the design is done, parasitic resistances and capacitances are extracted and used to calculate the actual path delay of the clock signal. This step verifies whether the chip meets setup and hold for all timing paths.

GDSII Creation (11)

The final design is converted to a GDSII file format, which is the one accepted by most foundries. The act of generating this file is usually called “hardening” or “design hardening”. It is very common for a system to not be hardened all at once, but rather one design at a time. These are then integrated as macros into a bigger design, until the whole system has been converted.

The process of obtaining the final file containing all the information of the system is also commonly known as “Tapeout”.

2.3 Study on RISC-V

This section will present an introduction on processor ISAs, followed by an overview and comparison of the existing ones. The focus will be on RISC-V and will attempt to explain why it was chosen for this work.

2.3.1 Instruction Set Architecture (ISA)

An Instruction Set Architecture (ISA) is not an actual physical part of a computer, but rather an abstract model that defines how the CPU is controlled by the software. It acts as an interface between software and hardware, specifying both what the processor can do, as well as how it gets done. It also defines the type of instructions the CPU can make and what parameters it takes when performing operations, as well as tells us where the registers are, and what the input/output interface looks like for a given system.

An ISA is not to be confused with a microarchitecture, the latter being the implementation of the ISA in a processor. A single ISA can have different microarchitecture implementations. This is a very positive aspect as different chips can use the same ISA and therefore run the same programs, while offering different trade-offs in price, performance, and power-consumption. It also allows software and hardware development to be separated from each other. An additional advantage is that it can also be extended by adding further instructions and support for longer addresses and data types. Nevertheless, any extended version of an ISA should be able to run machine code executable by previous versions.

However, no ISA design is perfect. Due to the need to maintain compatibility with old software, early design choices, whether bad or simply outdated, can have lasting impacts into the future, limiting design choices for new CPUs.

2.3.2 CPU Architecture Types

There exist two different architecture types nowadays. One being Complex Instruction Set Computer (CISC) and the other Reduced Instruction Set Computer (RISC). This section will present a comparison between the two while later subchapters will focus more on the RISC architecture, as that is the one used in RISC-V.

CISC is the older architecture, which was invented in the 1970's. At the time, memory was expensive, so having fewer instructions was advantageous. In response to this, the processors at the time were designed to perform less, but longer and more complex instructions (hence the name CISC) to reduce program size and consequently memory usage.

In contrast, the later invented RISC implements a design based on simple and fast instructions. These are small, usually a word only and most can be completed in a single machine cycle. This enables the use of the pipelining technique, which is widely used to speed up RISC based machines [5].

RISC vs CISC

The following Table 2.2 summarizes the main differences between RISC and CISC instruction sets.

Table 2.2 – RISC and CISC comparison.

Parameter	RISC	CISC
Instruction size	1 word	1 or more words
Execution time	1 clock cycle*	1 or more clock cycles
Efficiency	High	Low
Code	Complex	Simple

*Provided a Harvard architecture is used.

2.3.3 Evolution of RISC-V

Origin of RISC-V

RISC-V, like its name implies, is the fifth iteration of a RISC based ISA developed at UC Berkeley. The first iteration, RISC-I, was developed between 1980-1984 by a team led by David Patterson (who then coined the term RISC). It emphasized on its simpler instruction set, fixed length opcodes, load/store architecture and higher number of registers [6].

RISC-II (1983) followed and introduced condensed 16-bit instructions that would expand to 32-bit. Both RISC-III (1984) and RISC-IV (1988) were designed by Patterson's students to run Smalltalk and Lisp, respectively.

It was only in May 2010 that work on RISC-V begun as part of the Parallel Computing Laboratory, a five-year project to advance parallel computing, funded by Intel and Microsoft. Although it wasn't the objective, RISC-V was developed by the need of a highly flexible and extensible base ISA, on which they

could build their research efforts around. It also made sense to create a new ISA from an academic perspective, due to the complexity and proprietary aspects of commercial ones [7].

In Figure 2-18 is represented the evolution of Berkeley's ISA along the years.

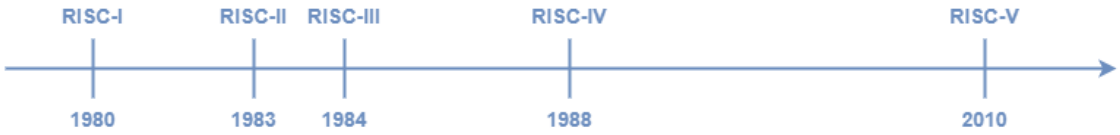


Figure 2-18 – RISC iterations over the years.

From 2010 to 2016, RISC-V continued to be polished and tested until the team at UC Berkeley felt comfortable with its final result. By 2015, interest in RISC-V continued to grow so much so that over 40 companies attended its first workshop that year. This event was what sparked the creation of the RISC-V Foundation [8], a non-profit corporation with the objective to build an open and collaborative community based on the RISC-V ISA.

Since then, the RISC-V ecosystem has been evolving rapidly, with new extensions, hardware applications and software compatibility being added every year, like shown in Figure 2-19.

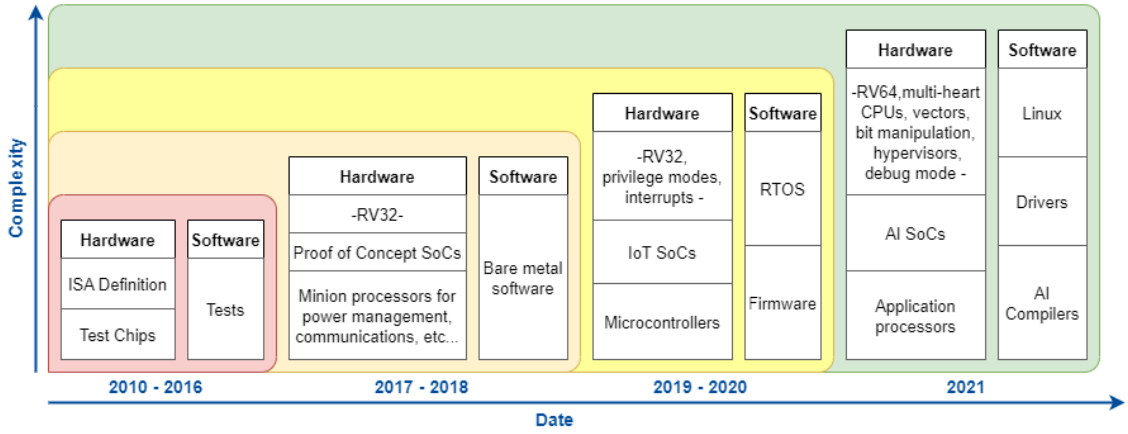


Figure 2-19 – Industry innovation on RISC-V.

Future of RISC-V

The areas RISC-V is expected to have the most impact going on are the fast-growing IoT, 5G and AI. All of these require a lot of computational power and often have energy efficiency constraints, which calls for specialized hardware. As RISC-V was designed for extensibility and is open source, it allows for extensive hardware customization which all these areas need.

2.3.4 Overview of existing ISAs

How much an ISA affects performance, power and energy is a debate that started long before the appearance of RISC-V, with the 90s RISC vs CISC debate. Although there is still not a definitive answer, some studies suggest the choice of ISA is irrelevant, [9,10] .

While the ISA might not have a direct impact in the overall performance, it has other effects such as:

- Implementability: facility of designing efficient, high-performance implementations.
- Programmability: how easy it is for the compiler to generate code.
- Compatibility: capability of maintaining support for older programs as technology evolves.
- Scalability: upgradability of the ISA.

A few other ISAs will also be studied in the following sections.

MIPS

MIPS's (Microprocessor without Interlocked Pipelined Stages) main problem is the poor flexibility of its instruction set, leaving very little room for different hardware implementations. The encoding leaves no available opcode space, so when designers decided to add a compact encoding, they had to create a different mode with a separate instruction set, [11] .

As of March 2021, the development of the MIPS architecture has ceased. The company then responsible for MIPS, Wave Computing, has joined the RISC-V Foundation and their future designs will be based on the RISC-V ISA.

SPARC

SPARC's (Scalable Processor Architecture) most distinctive feature is the register window. While it's convenient that a part of the stack can be stored in registers, a lot of these were not actually used. They are also expensive to implement and don't work for floating point [12].

A later version of SPARC, (v9) completely scrapped the register window idea and implemented 32 global registers instead.

POWER

POWER (Performance Optimization With Enhanced RISC) is an instruction set developed mainly by IBM [13]. Although it's not a simple architecture, it's still not as complex as ARM. In POWER, each register belongs to a functional class, with most of the instructions in a certain class using only registers that belong to it. As such, only a few instructions pass data between different classes. However, interaction between different classes may lead to implementation-dependent delays.

The architecture also features other specific instructions, such as generating a random number in a register or multiple store and loads. This really boosts POWER's performance, but also increases its complexity. Because of this, its applications are mainly in supercomputers and microcontrollers.

Its complexity and highly specific applications became the biggest hinderance to the formation of a wide developer community. This has improved in recent years, however. With the ISA becoming public, several open-source projects and FPGA implementations have surfaced.

This makes POWER an ISA worth considering, and possibly a competitor to RISC-V.

RISC-V vs ARM

ARM is currently the most successful RISC ISA available and is dominant in microcontrollers microprocessors and mobile systems [14]. It is without a shadow of a doubt RISC-V's biggest rival. The main differences between the two ISAs have been highlighted in the Figure 2-20 below.

ARM	RISC-V
Proprietary ISA Must pay fees to use	Open-source ISA No fees to use
Large community Has many libraries for different platforms	Relatively new community Still growing in terms of support for different platforms
Small number of vendors	Various open-source cores available, as well as commercially licenced ones
Has many hardware systems to help designers incorporate their CPUs	Very little support for hardware design (As it's still new)
Being proprietary, runs the risk of being blocked by governments	It's open-source, and as such, is available to anybody
Initially not designed to use custom extensions	Designed from the start to be highly scalable and easy to add extensions to
Very mature ISA	Relatively new ISA Still missing some special extensions
Very Complex for a RISC ISA Hard to learn	Simple and short base ISA Extremely easy to learn

Figure 2-20 – Main differences between ARM and RISC-V ISAs.

From the image it becomes obvious that RISC-V's advantages against ARM lie in the freedom it provides its users with. This results from it being open-source, but also from the fact that it was designed to be easily upgraded. ARM, on the other hand, gains from its long presence in the market and very mature and developed environment.

The main disadvantages of RISC-V come from it still being relatively new. However, it is evolving at a very rapid pace, as communities worldwide are implementing RISC-V and creating new extensions and hardware design environments. This is another benefit of open-source.

Several studies have been made to compare implementations of these two ISAs. Parameters like energy efficiency and execution times have been a common target of comparison. Such analysis can be found on [15] and [16].

According to SiFive, one of the leaders in RISC-V chip development, their current highest performance processor, the P650, is comparable to ARM's Cortex-A77 while maintaining a significant

performance-per-area advantage [17]. The company, founded in 2015, has quickly set itself has an alternative to ARM.

Although their highest performance chips are still not as good as ARM's, they've been closing the gap in the last few years. Table 2.3 presents a comparison between SiFive's chips and its equivalent from ARM, including announcement dates as well as their corresponding time difference.

Table 2.3 – SiFive and ARM chip comparison.

SiFive Model	Announcement Date	ARM Equivalent	ARM Equivalent Announcement Date	Time Difference
U74	October 2018	Cortex A53	October 2012	6 years, 0 months
U84	October 2019	Cortex A72	April 2015	4 years, 6 months
P550	June 2021	Cortex A75	May 2017	4 years, 1 month
P650	December 2021	Cortex A77	May 2019	2 years, 7 months

Figure 2-21 shows a graphic representation of such difference at the time of SiFive's new chip announcement. It also attempts to estimate the rate at which SiFive is catching up to ARM.

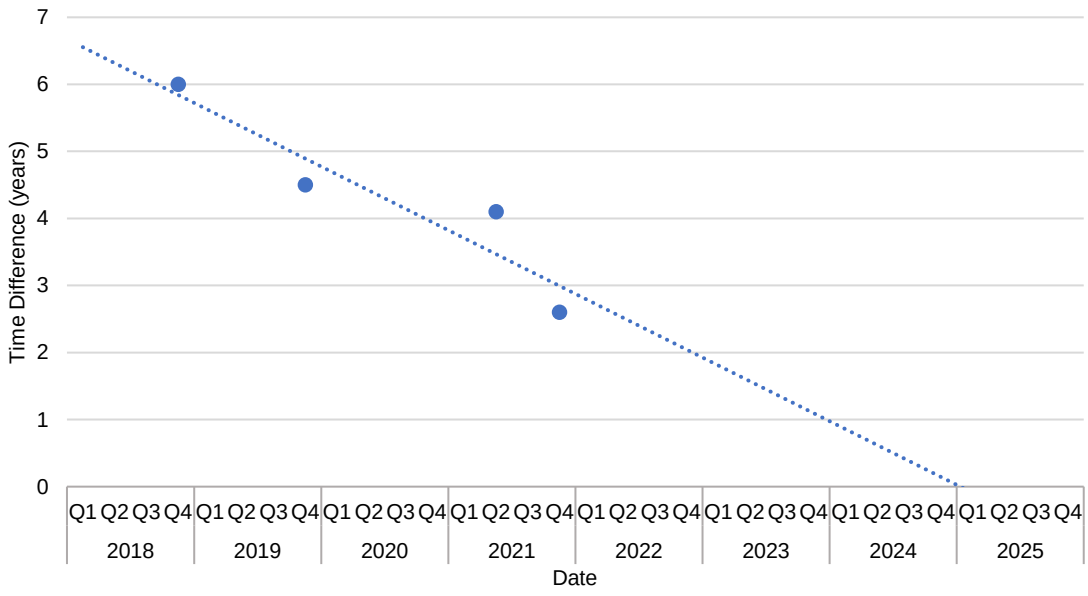


Figure 2-21 – Time difference between ARM's and SiFive's chips.

Assuming a linear progression, and that the tendency continues, SiFive's chips should catch up to ARM's around Q1 2025.

2.3.5 RISC-V Implementations

This section will attempt to summarize the state of the RISC-V technology in the following two main areas:

- Development boards.
- Smartphones.

Development Boards

HiFive Unmatched is a RISC-V development board created by SiFive. It is powered by their U740 SoC and is capable of running Linux [18].

It's a PC-oriented board, which allows developers to experiment with RISC-V. Its price comes at around 600€, which is quite high, but justifiable considering its specs, the limited production, and the fact that it's currently the best available RISC-V option. A performance test followed by a Raspberry Pi 400 comparison can be found in [19]. Their results show that the HiFive Unmatched is far outclassed by the Pi. This might be, however, due to the benchmarks used having x86/x86_64 and ARM/AArch64 optimizations and little to no RISC-V ones. This makes it unclear how much of Raspberry Pi's advantage comes from hardware and how much from RISC-V's still evolving Linux support.

Microcontrollers and IoT are probably the areas RISC-V is most developed in. There are already several microcontroller boards available in the market from different providers. The Table 2.4 below presents some of the already existing options.

Table 2.4 – RISC-V Microcontroller boards

Provider	Board	Processor	Uses	Price
SparkFun	RED-V RedBoard [20]	FE310-G002	Microcontroller	~35€
SparkFun	RED-V Thing [21]	FE310-G002	Microcontroller	~26€
GroupGets	LOFIVE-R1 [22]	FE310-G002	Microcontroller	~27€
Sipeed	Maixduino [23]	RISC-V Dual Core 64bit with FPU (400MHz)	Microcontroller AI + IoT	~30€

SparkFun offers both the RED-V RedBoard which comes in an Arduino Uno form factor, and RED-V Thing, a smaller, 23 x 60 mm board. Both feature the Freedom E310 SoC.

LOFIVE-R1 is a 38 x 18 mm microcontroller board by GroupGets. Its very small size yet fast CPU distinguishes it from others.

Lastly, there's Sipeed's Maixduino, designed to run AI systems. It has built-in WiFi and Bluetooth in the form of an onboard Esp32 module as well as a camera and LCD connectors. Its relatively low price and wide peripheral compatibility makes it a great development board for the ever-growing AI and IoT systems.

Smartphones

As of writing, there is currently no RISC-V smartphone available yet. However, in early 2021, T-Head, Alibaba's semiconductor division, showed a fully functional version of Android 10 running on a RISC-V processor. Later, on November 8th, 2021, Sipeed also announced they successfully ran Android 10 on a 64bit RISC-V chip (C910). The company hopes to have a developer-oriented smartphone out by 2022/2023.

3 PROPOSED OPEN-SOURCE TOOLSET

This chapter presents the open-source tools and opportunities chosen for the proposed design flow described in chapter 4.

3.1 Openlane

Openlane is an automated RTL to GDSII flow that is based solely on open-source components. Its objective is to have a no-human-in-the-loop flow with less than 24 hours turnaround, at eventually no loss of PPA (Power-Performance-Area). By giving it a RTL design in the form of Verilog, and selecting the desired PDK, the Openlane carries out the physical design process, previously described in chapter 2. It achieves this by utilizing various open-source tools in succession, each dedicated to a specific part of the flow. In essence, Openlane is nothing more than some TCL and Python scripts stitching together specialized tools inside a docker container. A summary of all of these tools and their respective function within the system can be observed in Table 2.1Table 3.1.

Table 3.1 – Openlane Tools.

Stage	Tool	Function
Synthesis	Yosys	Performs RTL synthesis.
	abc	Performs technology mapping.
	OpenSTA	Performs static timing analysis on the resulting netlist to generate timing reports.
Floorplan and PDN	init_fp	Defines the core area for the macro as well as the rows (used for placement) and the tracks (used for routing).
	ioplacer	Places the macro input and output ports.
	pdn	Generates the power distribution network.
	tapcell	Inserts welltap and decap cells in the floorplan.
Placement	RePLace	Performs global placement.
	Resizer	Performs optional optimizations on the design.
	OpenDP	Performs detailed placement to legalize the globally placed components.

Stage	Tool	Function
CTS	TritonCTS	Synthesizes the clock distribution network (the clock tree).
Routing	FastRoute	Performs global routing to generate a guide file for the detailed router.
	CU-GR	Another option for performing global routing.
	TritonRoute	Performs detailed routing.
	SPEF-Extractor	Performs SPEF extraction.
GDSII Generation	Magic	Streams out the final GDSII layout file from the routed def.
	Klayout	Streams out the final GDSII layout file from the routed def as a back-up.
Checks	Magic	Performs DRC Checks & Antenna Checks.
	Klayout	Performs DRC Checks.
	Netgen	Performs LVS Checks.
	CVC	Performs Circuit Validity Checks

Even though its automated, users are allowed to set flow variables to have full control of the whole process. These can be constraints or objectives of the design, or simply different options or methods of a particular tool. The list of all available variables can be found in the Openlane docs [24].

Aside from these, Openlane also provides two exploration methods, namely synthesis and design. The first uses different strategies for logic synthesis and technology mapping to optimize for things like area or delay. The second runs multiple iterations of the flow, with different parameters every time, all to way to the detailed routing phase, skipping all further verification. This allows users to quickly experiment with and optimize their designs.

Figure 3-1 gives a high-level view of the Openlane flow architecture. Note that although it appears in the flow, the DFT step is not yet integrated.

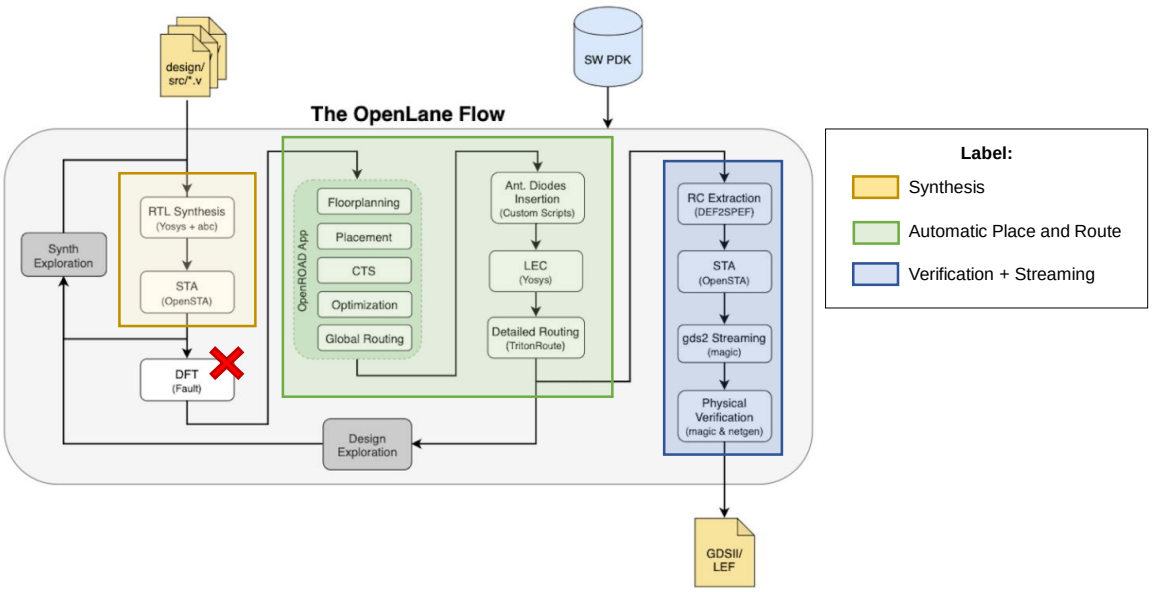


Figure 3-1 – Openlane flow architecture.

Macro Hardening Tutorial

To give a better understanding on how the flow works on a practical level, a small tutorial is offered in this section. Later, in Chapter 4, an example on how to integrate a macro into Caravel is offered.

From now onwards, it is assumed that the framework and all other necessary tools have been successfully installed and configured.

Since Openlane is configured inside a docker container, to access its environment we need to first to run the command

```
make mount
```

in a terminal shell inside the main directory.

There is a folder in the root directory called “designs”. Like the name implies, this is where all already available designs reside, and where the user should build their project in. Every design folder must include at least a source folder with all Verilog files (src), and a configuration file with all flow specifications (config.tcl). A new basic project with these can be created using the command

```
flow.tcl -design <design_name> -src <design_folder/design.v> -init_design_config
```

where <design_name> is the project name and <design_folder/design.v> is the path to the Verilog source.

The generated configuration file is demonstrated below. Right now, it only contains the bare minimum to harden a design, but additional specifications can be added by the user.

In case the project contains more than one Verilog source, they must be listed in the variable “VERILOG_FILES”.

```
# User config
set ::env(DESIGN_NAME) example

# Change if needed
set ::env(VERILOG_FILES) [glob $::env(DESIGN_DIR)/src/*.v]

# Fill this
set ::env(CLOCK_PERIOD) "10.0"
set ::env(CLOCK_PORT) "clk"

set filename $::env(DESIGN_DIR)/$::env(PDK)_$::env(STD_CELL_LIBRARY)_config.tcl
        if { [file exists $filename] == 1} {
          source $filename
        }
```

Before running the flow, all modules in Verilog source files should be updated to include power pins. Although this isn't strictly necessary, it helps with the power routing and LVS verification, so it is highly recommended.

```
Module example (
`ifdef USE_POWER_PINS
        inout vccd1,
        inout vssd1,
`endif
input a,
output b
);
```

"USE_POWER_PINS" is set by Openlane when hardening a design. With the use of a conditional compilation (`ifdef), the edited files can still be used in the same way in simulations and FPGA implementations by not defining this flag.

The first thing that should be done when attempting to harden a design is to do a synthesis exploration. Like previously mentioned, this will perform multiple synthesis of the design while prioritizing and monitoring different variables. In the case of Openlane, these variables are the gate count, the total area used, and delay. This exploration can be run with the command

```
flow.tcl -design <design_name> -synth_explore
```

If this happens to be the first run of the project, Openlane will automatically create a "runs" folder and output the results inside it. In this instance, the flow output is a HTML dashboard that includes all available synthesis strategies and their respective outcomes. It can be found in the reports/synthesis directory of the run.

Figure 3-2 shows an example of the dashboard for a PicoRV32, a RISC-V processor core.

Best Area	Best Gate Count	Best Delay
128512.0	15603.0	4712.38
AREA2	AREA1	AREA3

Strategy	Gate Count	Area (um^2)	Delay (ps)	Gates Ratio	Area Ratio	Delay Ratio
DELAY0	22877.0	195935.42	6640.63	1.466	1.524	1.409
DELAY1	23182.0	192639.75	6796.72	1.485	1.499	1.442
DELAY2	23110.0	192483.36	6360.84	1.481	1.497	1.349
DELAY3	22363.0	190936.88	6556.21	1.433	1.485	1.391
DELAY4	16383.0	145752.28	7941.18	1.049	1.134	1.685
AREA0	15700.0	131203.33	8867.82	1.006	1.02	1.881
AREA1	15603.0	129993.42	9531.27	1.0	1.011	2.022
AREA2	15707.0	128512.0	9967.29	1.006	1.0	2.115
AREA3	22192.0	154176.61	4712.38	1.422	1.199	1.0

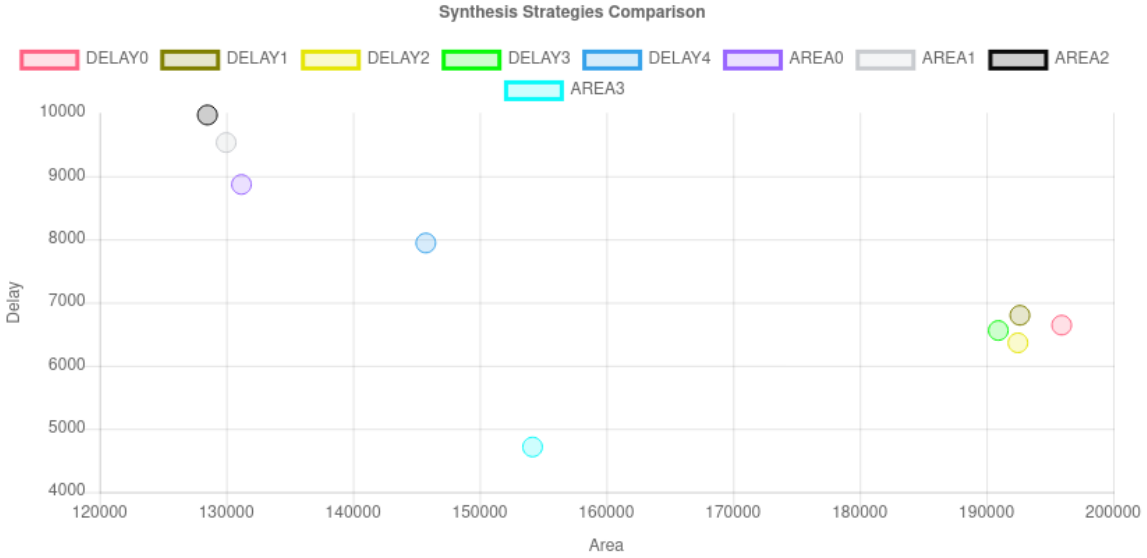


Figure 3-2 – Synthesis exploration dashboard.

From this analysis, the user can choose the appropriate strategy for their needs. On the one hand, if the objective was to minimize area at all costs, then "AREA2" is the best strategy. On the other hand, "AREA3" not only is the best for reducing delays, but also has decent area usage.

After deciding on a synthesis strategy, it is a good idea to run a design exploration with a few different flow variables. Table 3.2 covers the most impactful of them and explains how each affects the flow.

Table 3.2 – Important flow variables for design exploration.

Variable	Step	Description
SYNTH_STRATEGY	Synthesis	Sets the strategy to use during synthesis.
SYNTH_MAX_FANOUT	Synthesis	Maximum number of loads a single cell can drive.
FP_CORE_UTIL	Floorplan	Defines the percentage of the die that is occupied by design instances.
FP_ASPECT_RATIO	Floorplan	Ratio between height and width of the die.
PL_TARGET_DENSITY	Placement	Defines how close together cells are.

The previous variables are very useful when trying to find the optimal configuration of a design, especially to minimize area usage.

Design exploration is run with the help of a python script, and can be started with the command

```
python3 run_designs.py --regression <my_configuration.config> --threads <x> <design>
```

where <my_configuration.config> is the file containing the flow variables to sweep, <x> is the number of threads to run in parallel, and <design> is the project name.

An example of an exploration configuration file can be found below.

```
SYNTH_MAX_FANOUT=(3,4,5,6,7)
FP_CORE_UTIL=(30,40,50,60)
PL_TARGET_DENSITY=(0.35,0.45,0.5,0.55,0.6)
FP_ASPECT_RATIO=(0.5,0.75,1,1.5,2)
```

Each variable's values should be comma-separated and have no spaces in between. If a configuration conflict occurs between this file and config.tcl, this takes priority.

In the situation of the previous example, Openlane would run 625 iterations of the flow to complete all possible combinations of the defined parameters. Usually, it is not wise to do such big regressions at once, but rather to run more and smaller ones. Since it's very likely that some of the assigned values never result in a valid implementation, trying them with other parameters is a waste of time. For example, if the tools cannot harden a design with 60% core utilization no matter what, then the previous configuration would be wasting 125 runs. More iterations also mean more disk usage, so resource-constrained machines might be limited to smaller regressions. Otherwise, when time and computational power aren't an issue, this approach is the simplest.

After finishing the process, the results are outputted into a CSV (coma separated values) file in the "regression_results" directory. Each individual run is also available in the "runs" folder of the project. An analysis of CSV data is given in the next chapter 4.

Aside from these, there are other important variables to set that do not make sense to experiment with in the design exploration. Those are presented in Table 3.3.

Table 3.3 – Most relevant flow variables.

Variable	Step	Description
FP_SIZING	Floorplan	Can be set to "absolute" to use a defined die area instead of using FP_CORE_UTIL.
DIE_AREA	Floorplan	If FP_SIZING is set to "absolute", specifies the die area in mm.
FP_PDN_VOFFSET	Floorplan	The offset of the vertical metal stripes in metal 4.
FP_PDN_HOFFSET	Floorplan	The offset of the horizontal metal stripes in metal 5.

Variable	Step	Description
FP_PDN_VPITCH	Floorplan	The pitch of the vertical metal stripes in metal 4.
FP_PDN_HPITCH	Floorplan	The pitch of the horizontal metal stripes in metal 5.
DESIGN_IS_CORE	Floorplan	Specifies whether the design is the core or a macro inside it.
FP_PIN_ORDER_CFG	Floorplan	Receives the path to the pin order configuration path.
VDD_NETS	Floorplan	Defines the power nets and pins to be used.
GND_NETS	Floorplan	Defines the ground nets and pins to be used.
RT_MAX_LAYER	Routing	Max layer to be used for routing.
ROUTING_CORES	Routing	Number of threads to be used by TritonRoute.
PDK	Misc	Specifies the PDK.
STD_CELL_LIBRARY	Misc	Specifies the standard cell library.
DIODE_INSERTION_STRATEGY	Misc	Defines the diode insertion strategy to be used.

When both explorations have been done, it is time to add the necessary parameters to the config.tcl file. Below is an example with most of the variables mentioned before.

```
set ::env(DESIGN_NAME) Example
set ::env(VERILOG_FILES) "\
        $::env(DESIGN_DIR)/src/Example.v \
        $::env(DESIGN_DIR)/src/Example2.v"
set ::env(CLOCK_PERIOD) "20.0" #nanoseconds
set ::env(CLOCK_PORT) "clk"
set ::env(SYNTH_STRATEGY) "AREA 3"
```

To finally run a design, we can use

```
flow.tcl -design <design_name> -tag <run_name>
```

where <run_name> is the generated folder's name. Although the tag isn't mandatory, it helps in organization.

At the end of the flow, the tools check for possible violations in the design. In case none are found, the flow succeeds, and the last results are saved in the "results/final" directory. Some violations like antenna and fanout don't fail the flow and only give a warning. If the flow fails, a reason is provided so that the user can adapt their design/configurations to meet criteria.

The final layouts can be observed in the ".mag" , ".def", and ".gds" files by using Magic, the OpenROAD Graphical User Interface (GUI) and Klayout, respectively. Figure 3-3 and Figure 3-4 show examples of them.

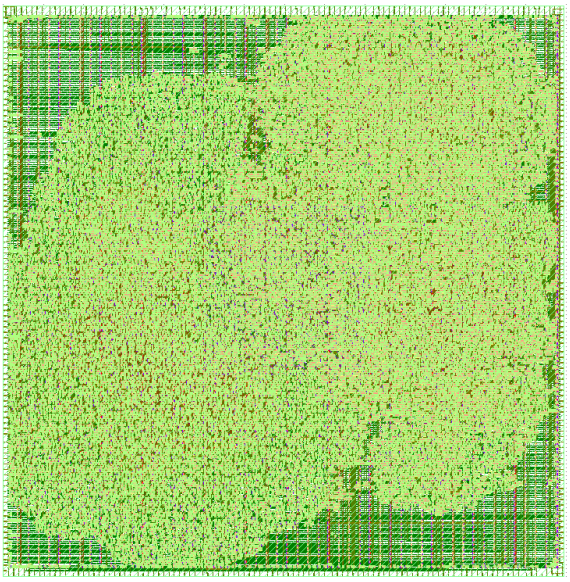


Figure 3-3 – PicoRV32 GDS view.

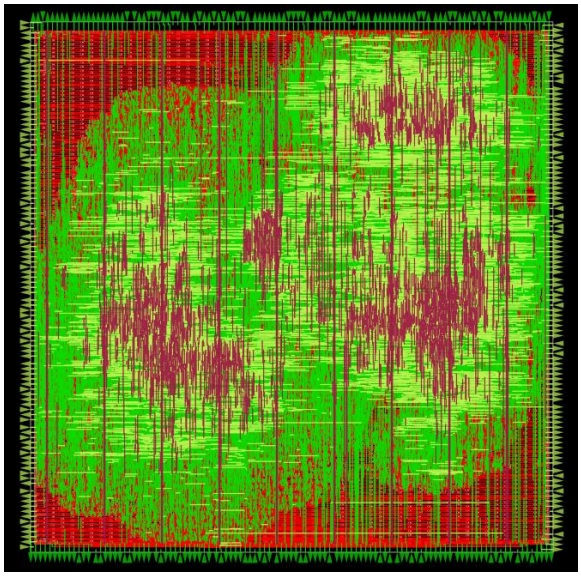


Figure 3-4 – PicoRV32 DEF view.

The OpenROAD GUI also provides users with other handy tools such as heat maps to better visualize data. To exemplify, Figure 3-5 offers the last layout but with Placement Density analysis activated.

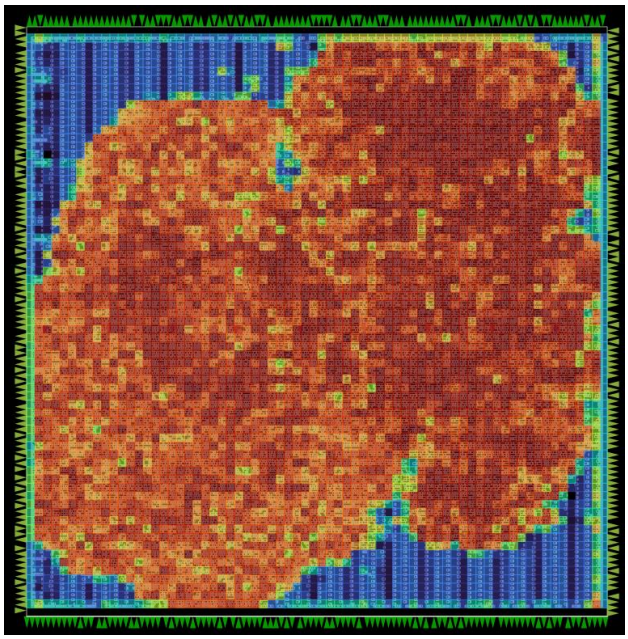


Figure 3-5 – PicoRV32 Placement Density Heatmap.

3.2 Open Process Design Kit

Originally developed by Cypress Semiconductor, the SKY130 process node is a 180nm-130nm hybrid technology. As a result of the collaboration between SkyWater and Google, the PDK is now completely open to the public.

Currently, the PDK is in an experimental stage, meaning it is not intended to be used for mass production, but rather for doing test chips and design verification.

The SKY130 Process node technology stack consists of the following, present in Figure 3-6:

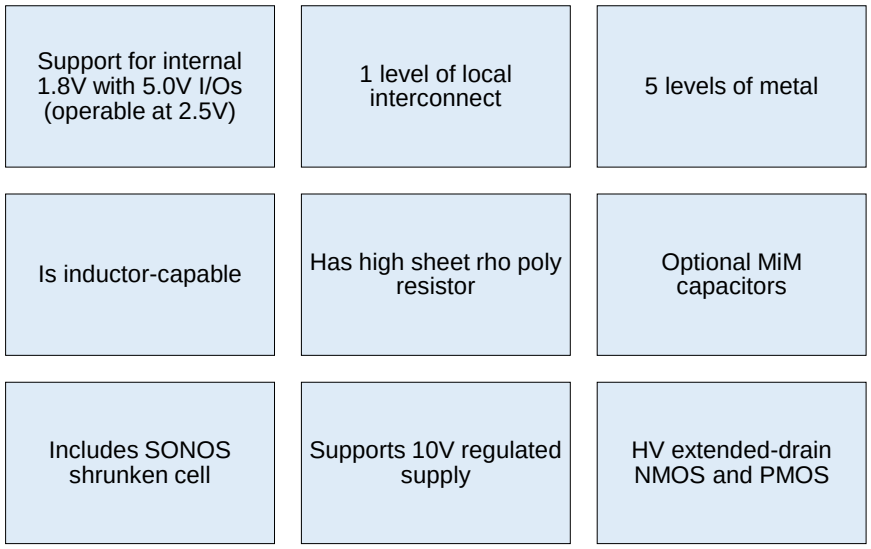


Figure 3-6 – Sky130 technology stack.

Although the 130nm process is considered an "old" technology, seeing as it dates back to 2001-2002, it is still widely used today for research and mixed signal embedded designs like IoT devices.

Currently, there are currently 7 standard cell libraries available, each with different perks. Extensive information on these can be found on the Skywater SKY130 docs, but a small summary is made in Figure 3-7 [25].

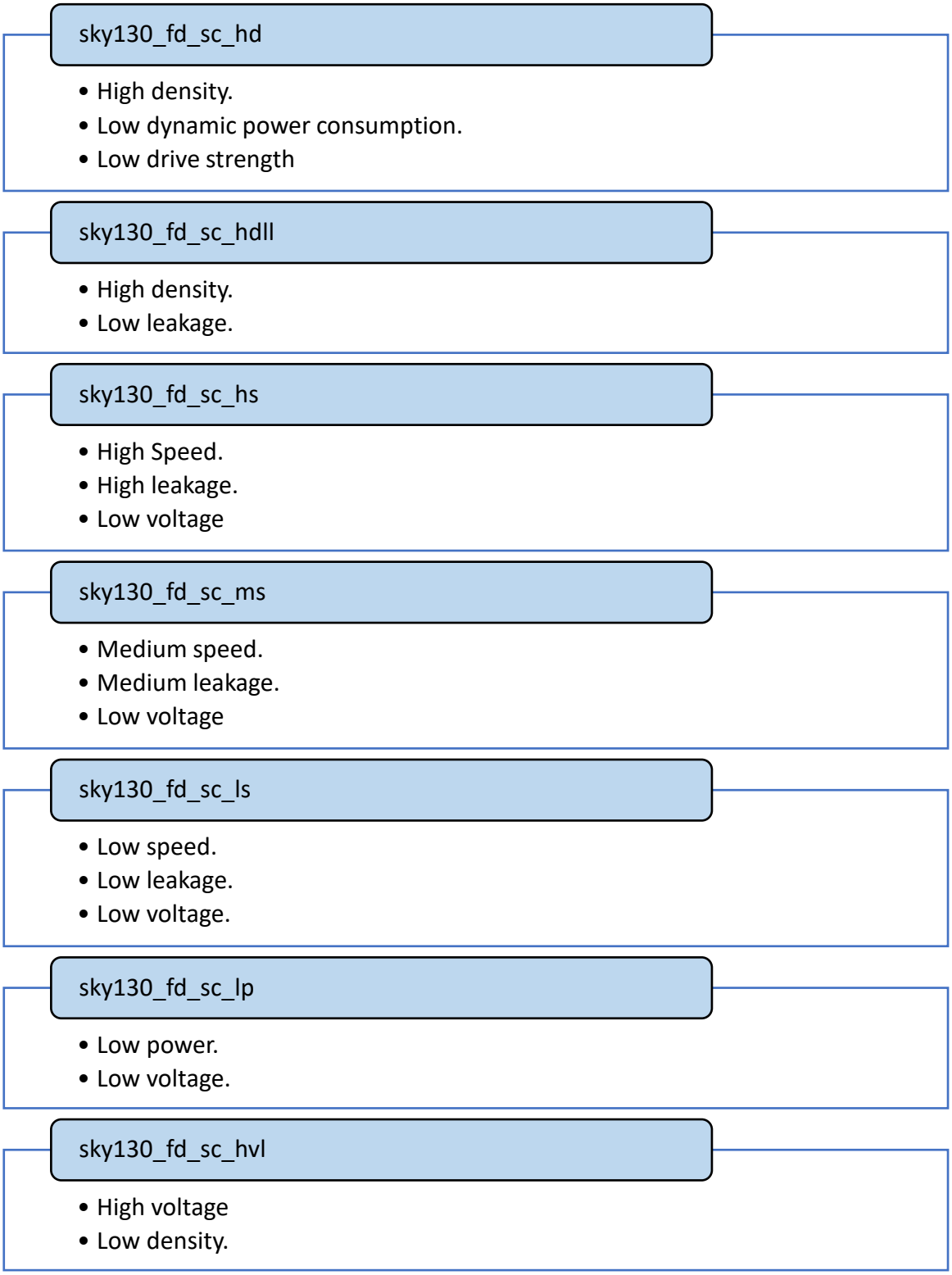


Figure 3-7 – SKY130 standard cell libraries.

Aside from SKY130, a new, also open-source 90nm process was announced by Google to be released in later 2022 [26].

3.3 Open MPW Shuttle Program

The open multi-project wafer (MPW) shuttle is a program that allows anyone to submit an open-source design and get it manufactured free-of-charge. Like seen in Figure 3-8, it is the result of a collaboration between Google (as the sponsor), Skywater Foundries (as manufacturer of the chips), and Efabless (as the manager of the project). It means to illustrate the potential of fully open-source IC design, from design tools and process design kits (PDK) to IPs.

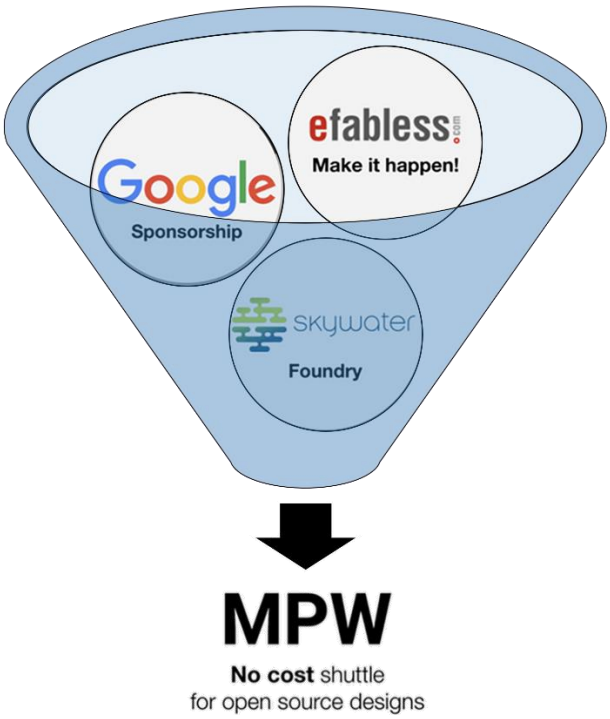


Figure 3-8 – Open MPW collaboration.

This program has its own special rules, which have been summarized below:

- The submitted project must be licensed has open-source.
- It must be posted in a git-compatible repository and publicly accessible, fully open, and reproducible.
- The design must use Skywater's 130nm Open PDK.
- All projects must use the Caravel harness and padframe.
- A precheck must be passed, including LVS and DRC checks.
- Each shuttle offers 40 available slots. If the number of submissions exceeds the these, a lottery will be drawn to determine the selected projects.

More information can be found on the Efabless' Open Shuttle Program page [27].

Every user is provided with a fixed 2.92mm x 3.52mm silicon area where they can insert their designs. To aid in implementing and testing these, Efabless developed Caravel, a SoC based on a RISC-V processor core that includes all the housekeeping functions to support user's designs. A graphical representation of design integration into caravel is given in Figure 3-9.

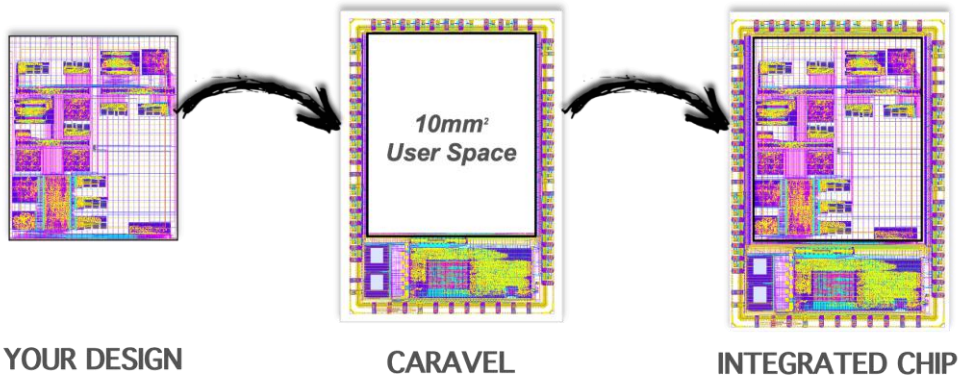


Figure 3-9 – Caravel design integration. Extracted from [27].

Its most important features are:

- VexRiscv core with debug port
- kB SRAM plus 1 kB of DFFRAM
- XIP SPI Flash controller
- UART, SPI and GPIO ports
- 128 port Logic Analyzer
- Counter / Timer
- 32-bit Wishbone bus extending to the user project area
- 6 user interrupts

All components of this SoC are also open-source. Further documentation is available in [28].

If a project is selected, the owner will receive 5 development boards and 50 packaged parts based on a WCSP (Wafer Chip Scale Package) package. An example development board can be found in the Figure 3-10 below.

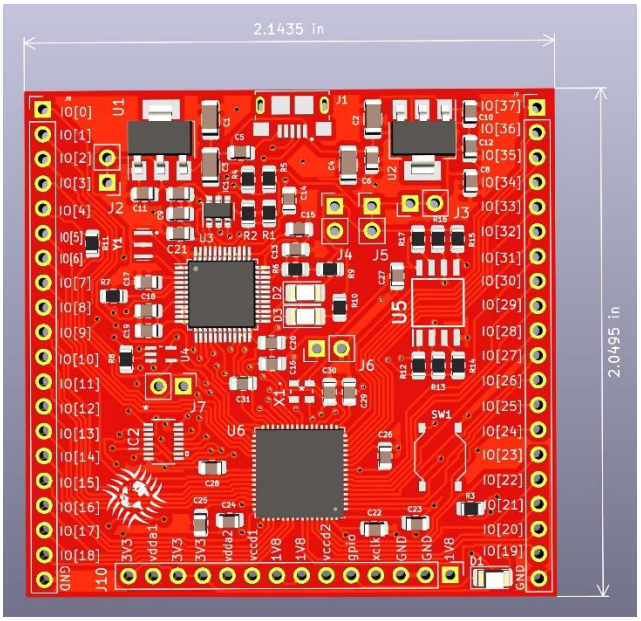


Figure 3-10 – Development board with user SoC.

4 DESIGN FLOW OF A SOC WITH RISC-V

This chapter documents the work done during this project. It starts by presenting the proposed design flow, then showing a practical example of how to employ it, divided by logic and physical design.

4.1 Proposed Design Flow

This subchapter presents the proposed Open ASIC design flow for RISC-V based SoCs. For simplicity purposes, the flow was divided into two sections, Logic Design and Physical Design. To better comprehend the particularities of each stage, the following Figure 4-1 and Figure 4-2 are given, to represent Logic and Physical design flows, respectively.

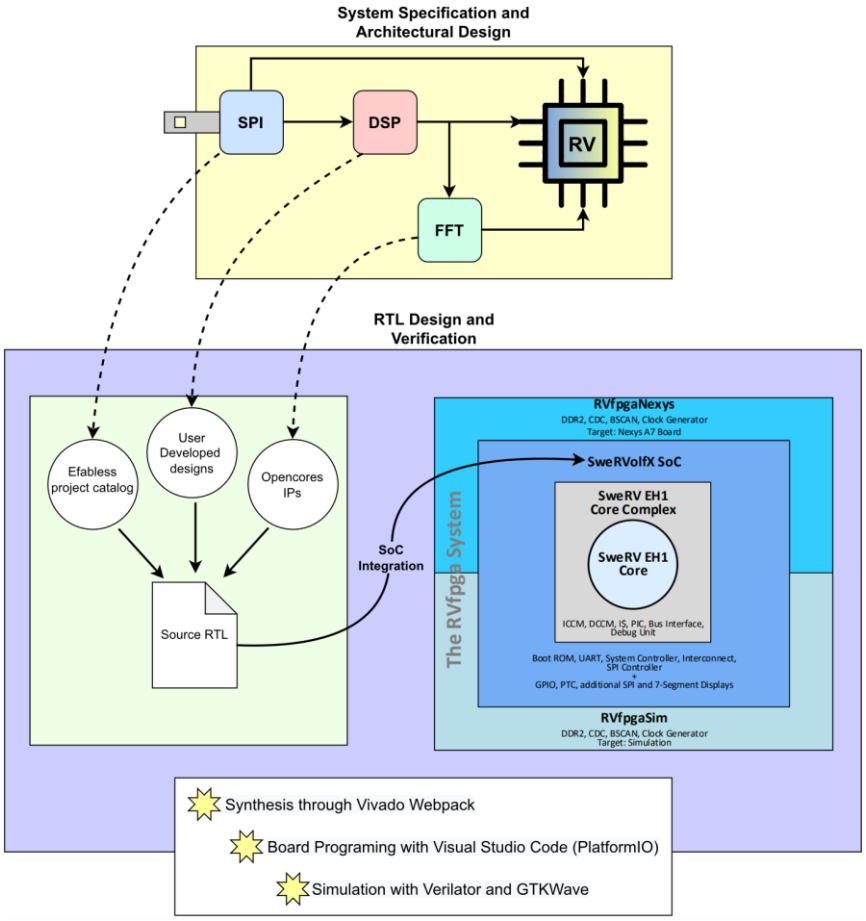


Figure 4-1 – Proposed logic design flow.

The focus of this flow is on the RTL design and verification. The system specification and architectural design phases are inherently more research focused and less tool dependent, so instead of enhancing the experience on them, an easier way to convert them to RTL is given. It proposes the use of already available open IPs, for example from Opencores and the Efabless project catalog. This drastically reduces design time since at most these designs only need to be adapted. In case a module doesn't already exist, the user can still develop it on their own and later publish in on the aforementioned platforms. Note that the modules shown in Figure 4-1 are purely examples.

To help in testing these designs, the SwervolfX SoC is used. Made by Imagination University Programme (IUP), it is an extended version of Swervolf, which was initially developed by ChipsAlliance [29]. It was created to run on a Digilent Nexys A7 FPGA, a perfect board for IoT development. The main reason it was chosen was for its excellent documentation by IUP, making it incredible for both testing and learning. The system gives full access to its users to modify it as they see fit, making it the perfect environment to test designs incorporated into a RISC-V based SoC.

This flow places high importance in FPGA testing, since it essentially validates that the system works in hardware. Therefore, the majority of design verification is done on it. Nonetheless, the flow still supports verification by simulation, and for that it uses Verilator and GTKWave. Verilator is used to generate traces from the Verilog files using C code as a stimulus, while GTKWave can observe them [30,31].

Being a Xilinx FPGA, the Nexys A7 uses Vivado to generate its bitstreams. However, Vivado is known for its clunky behavior, so users can instead program their RTL in Visual Studio Code (VSC) in case they prefer it. VSC is also used to flash software onto the FPGA by using the PlatformIO extension.

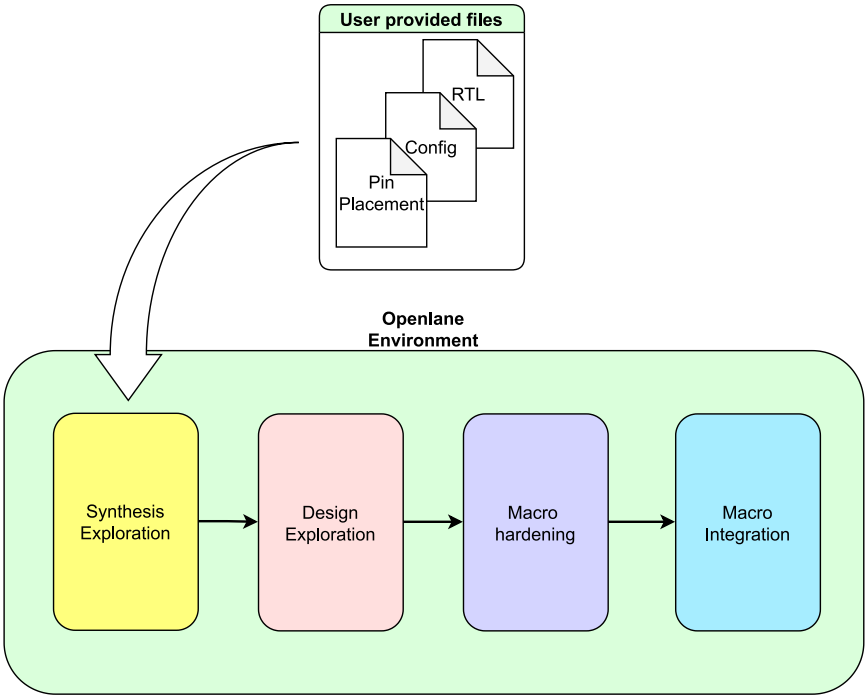


Figure 4-2 – Proposed physical design flow.

The physical flow is a lot simpler than the logic one at an architectural level, even though generally this stage includes more steps. This is due to the no-human-in-the-loop nature of Openlane. All the processes in it were condensed into the four phases seen in Figure 4-2. The first three have already been explained in the previous Chapter 3 while the fourth is the step at which the previously hardened macros are integrated into a single SoC. This was constructed with Caravel as the target SoC, but still works for any other chips.

4.2 Practical Example of the Proposed Design Flow

The rest of this section serves as an example of the design flow proposed above. During it, the first blocks of a low frequency application SoC will be developed. The objective is to eventually make a system similar to the one shown in [32]. Due to time constraints, it is impossible to fully adapt that SoC in its entirety, so only a selected number of modules will be incorporated for now. Figure 4-3 shows the architecture of the system and highlights which blocks will be included.

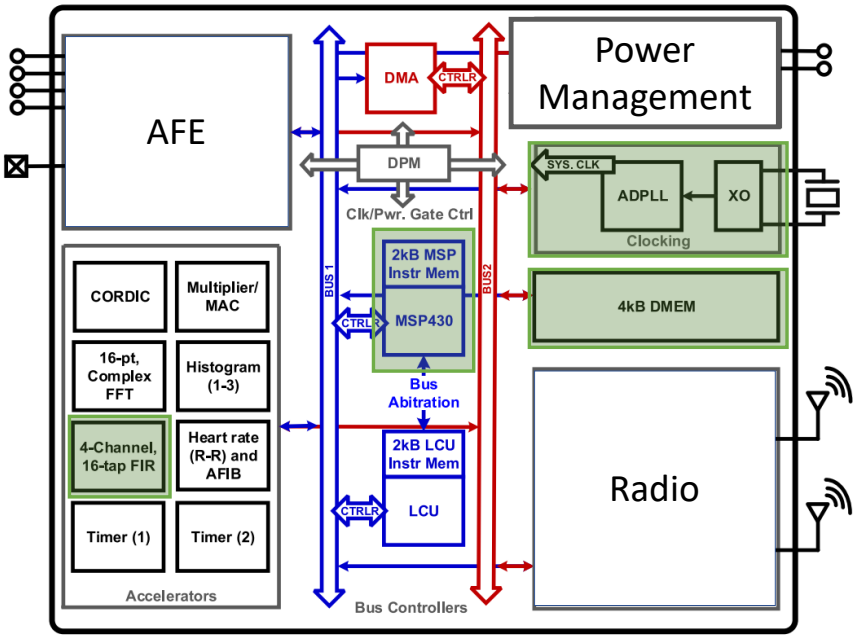


Figure 4-3 – Low frequency applications SoC. Adapted from [32].

Instead of a FIR filter, a biquad filter was be used. The biquad, short for biquadratic, is a second order IIR filter with two poles and two zeros. Therefore, it is a filter whose transfer function is the ratio between two quadratic functions, like seen in (4.1. These filters are highly configurable and require less resources than the FIR alternative but suffer from issues like phase distortion.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{4.1}$$

As for the processor, instead of the MSP430, a RISC-V core was used. For the FPGA testing phase, a SweRV EH1 was implemented, while for the actual chip a VexRiscv was used, since it's the one used in Caravel.

4.3 Logic Design

4.3.1 System specifications

The system to be developed targets low frequency applications such as ECG (Electrocardiogram) or Oximeters. It seeks to be fully open-source, so it will use a RISC-V processor and be manufactured with sky130 technology. It will use the Caravel SoC as a harness, so the chip dimensions will be 2,92mm x 3,52mm, with the packaging being a 6x10 WLCSP. Out of the 60 pins, 38 are GPIO. Figure 4-4offers an image of the package viewed from the bottom.

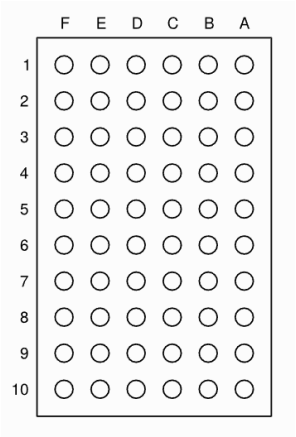


Figure 4-4 – Package viewed from the bottom.

To accelerate signal processing, a biquad filter will be added. It will feature 12-bit data and 16bit coefficients, so to be later interfaced with an AFE containing a 12-bit ADC. Multiple biquad filters can be cascaded to achieve higher order filters.

The system will work on 1.8V, with support for 3.3V for when analog blocks are incorporated and is designed to run at up to 50MHz.

4.3.2 Architecture design

After cleaning up the unincluded blocks from the previous Figure 4-3, we arrive at architecture depicted in Figure 4-5. Since one of the objectives is to integrate the design into Caravel, it's important to verify which modules from our system are already implemented in it. To help in visualizing this, both Figure 4-5 and Figure 4-6 (which shows the Caravel Architecture) mark these blocks.

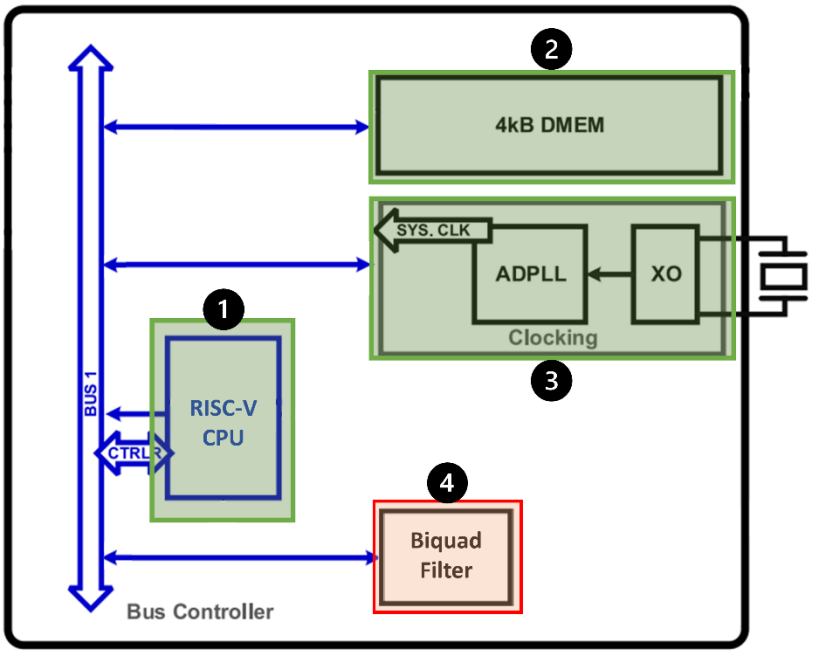


Figure 4-5 – Simplified SoC Architecture.

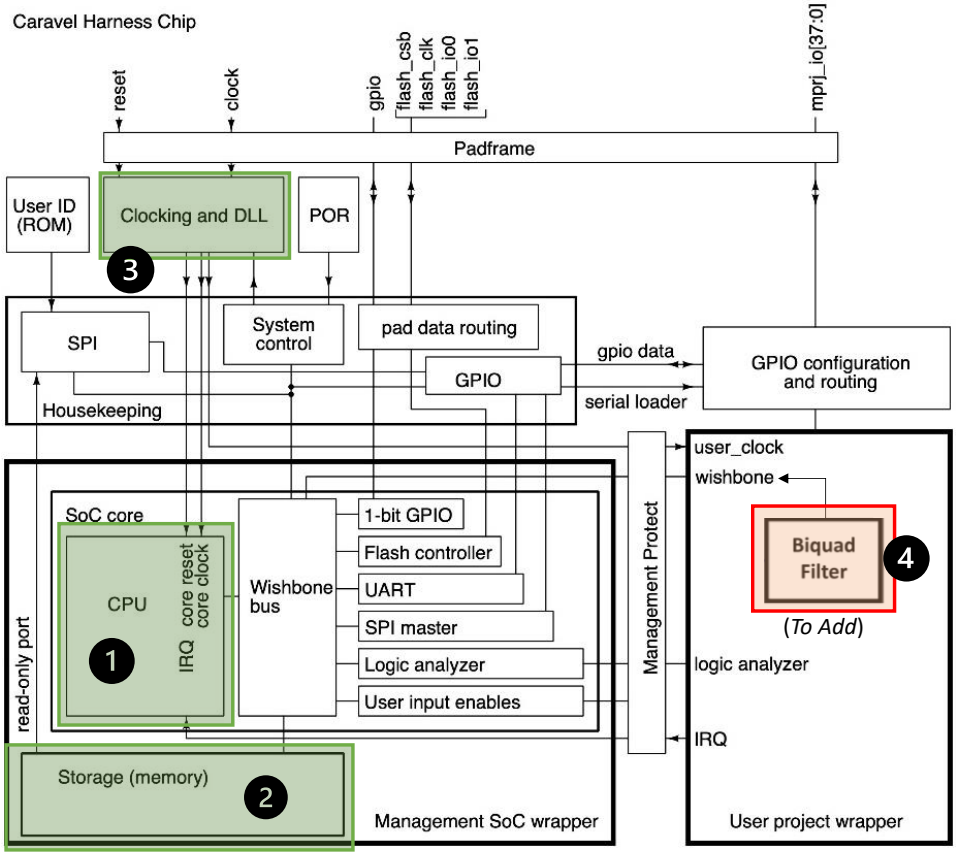


Figure 4-6 – Caravel SoC architecture.

With this analysis its evident that the only module from the system not yet included in Caravel is the biquad filter. As such, this is the only block that needs to be hardened and integrated using the Openlane tools. However, to properly test the biquad filter in a FPGA, it's important to have a system

similar to Caravel that could be synthesized in a FPGA. For this, the SweRVolfX SoC was used. This system was developed by Imagination University Programme [33] to be used in a FPGA and includes a RISC-V processor core, namely the SweRV EH1 which was designed and published by Western Digital. This serves as a great "canvas" to test peripherals in a RISC-V environment.

4.3.3 RTL Design and Design Verification

The HDL language chosen for this task was Verilog since it is the language adopted by the Openlane flow.

As previously stated, this project heavily incorporated FPGA testing during the Design Verification step. To help understand the verification flow used and the organization of this sub-chapter, the flowchart in Figure 4-7 was created:

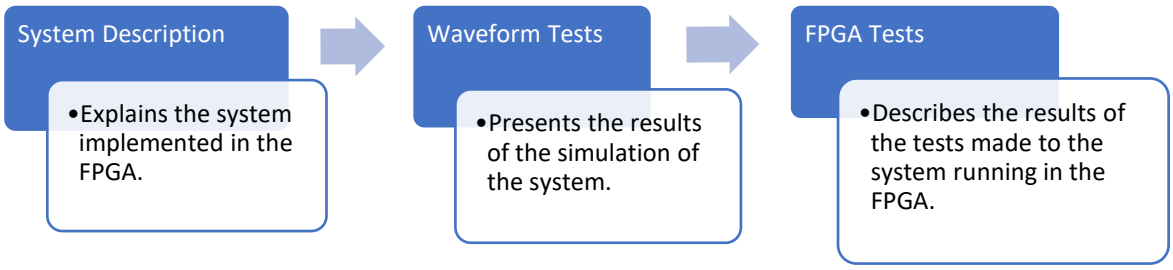


Figure 4-7 – Verification flow.

4.3.4 System description

To properly test the filters in conditions similar to those expected after fabrication, the following system (Figure 4-8) , written in Verilog, was implemented in a FPGA.

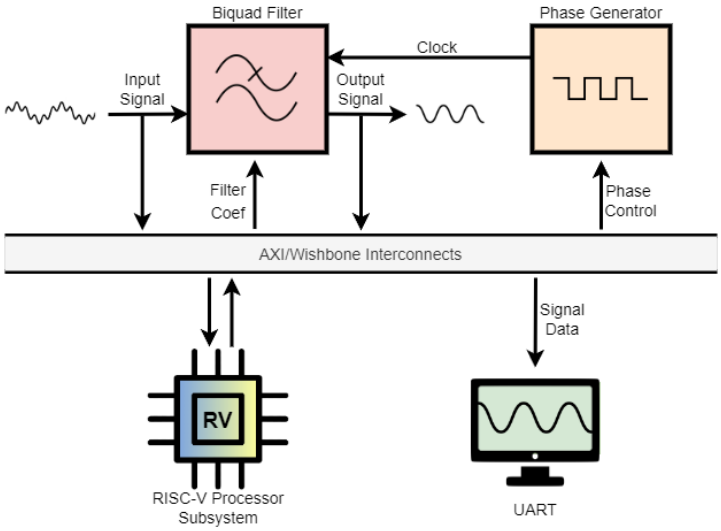


Figure 4-8 – Simplified FPGA system diagram.

It is composed by:

- A RISC-V processor subsystem, namely the SweRV EH1.
- A biquad filter module, which is effectively the unit under test.
- One phase generator that controls the sampling frequency of the filter.
- One Serial Monitor that receives filter data via UART.

In order to simulate a signal sampled from an ADC, an additional module was created to feed this signal to the filters (Figure 4-9). It achieves this result by utilizing a lookup table mapped with a test signal and outputting a new value each clock cycle (defined by the phase generator).

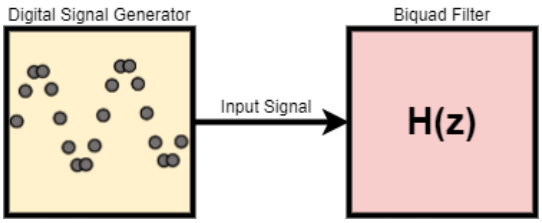


Figure 4-9 – Digital signal generator.

The filter's coefficients are registers mapped in memory. This means that it is possible to program them via software. To do that, Visual Studio Code was used, more concretely the PlatformIO extension.

4.3.5 Waveform Tests

C code was written to provide a stimulus to the system (Verilog files). Verilator was then used to generate the trace which was later observed using GTKwave.

Firstly, the digital signal generator module and the phase generator were tested. As seen in Figure 4-10, the module correctly outputs a new value (sig_o) every clock cycle from the phase generator (clock_out).

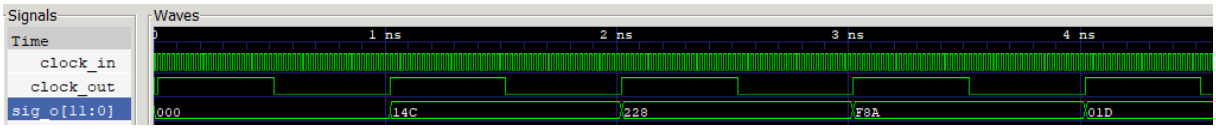


Figure 4-10 – Digital signal generator and phase generator test.

The second test's objective was to ascertain the programmability of the filter's coefficients, which in place also tests for proper wishbone communications. Its results are depicted in Figure 4-11.

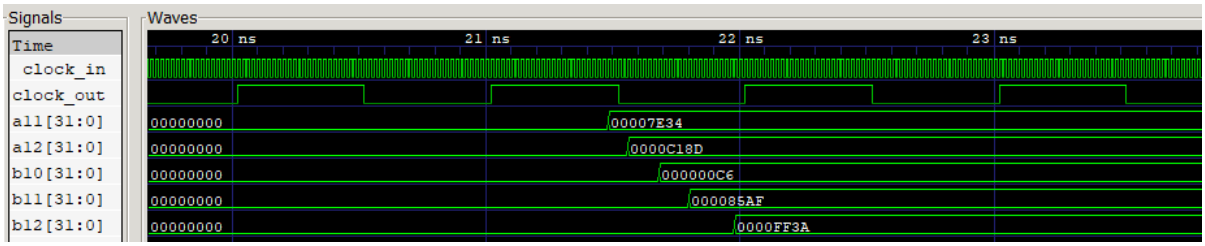


Figure 4-11 – Coefficients and wishbone test.

We can see from it that the coefficients are being set one after the other, this time synchronized with the clock signal “clock_in”, as it is the one used by the wishbone communication. This also tells us the RISC-V processor is working correctly since it could read software and act upon it.

Lastly, the filter was tested on whether or not it could correctly output a signal, given a certain input signal and coefficients (Figure 4-12) .

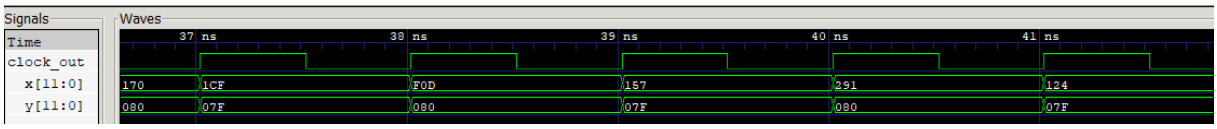


Figure 4-12 – Filter output test.

After confirming the correct behavior, one last full simulation was performed to observe the system as a whole (Figure 4-13) .

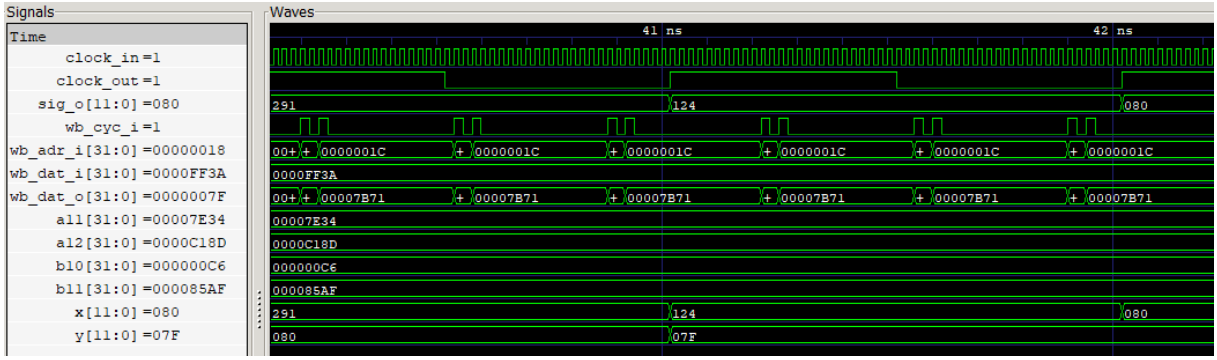


Figure 4-13 – Full system test.

4.3.6 FPGA Tests

The final step on the design verification was test the system running on a FPGA. The filter outputs were read and outputted to a serial monitor mainly for debugging. The data was also saved as a .csv file so it could be better analyzed in Excel.

This section will present two test scenarios. These tests serve to ascertain not only the correct behavior of the whole system, but also test the filter’s capabilities to represent different filter responses. During the tests, an input signal was injected into the biquad filter. Each test differs in input signal and therefore in the filter’s transfer functions as well. The tests are structured as followed:

- Setting – This explains the context of the test. What the goal of the test is and what the input signals are.
- Implemented Filter – Shows the implemented filter architecture as well as the filter’s transfer function and filter response.
- Simulated Filter Output – Presents the expected output of the filter.
- Tested Filter Output – The results from the FPGA test are shown here. These are also compared with the expected response.

The focus of these tests is not on finding the best transfer function for a specific input signal, but rather on whether the filters can correctly implement different functions.

Test 1

Setting

This is the simplest of the two tests. The input signal is composed by two signals one decade apart in frequency. The objective of this test is to determine whether the filter can separate these two signals. For the purpose of this test the considered frequencies were 10kHz and 100kHz and the higher frequency signal has 5 times the magnitude of the lower frequency one. The sampling frequency was 1MHz. The signals are present in Figure 4-14, Figure 4-15, and Figure 4-16.

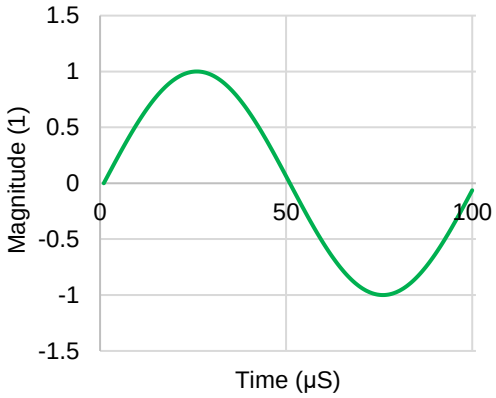


Figure 4-14 – Signal 1 (10kHz).

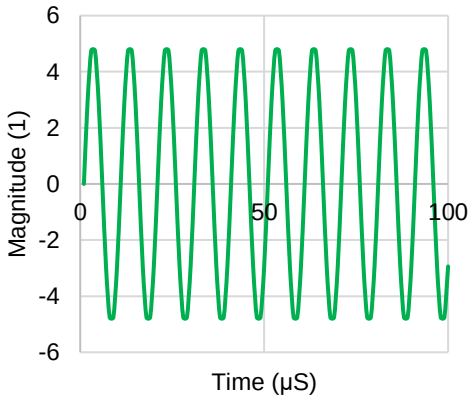


Figure 4-15 – Signal 2 (100kHz).

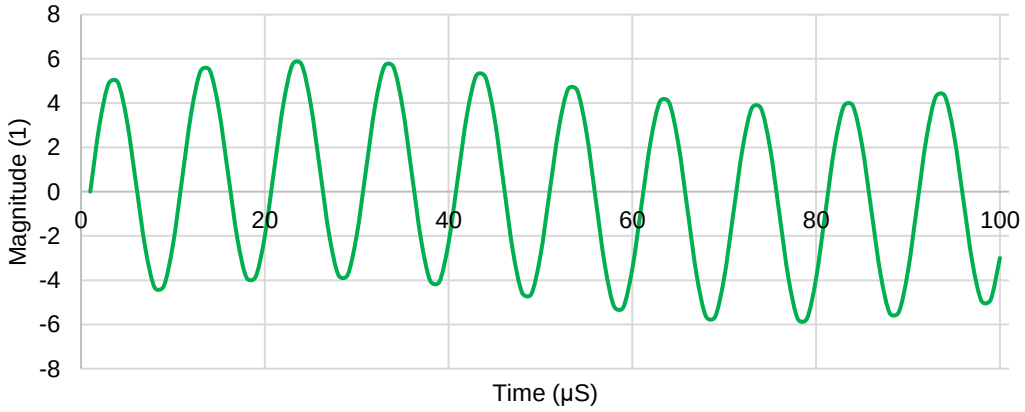


Figure 4-16 – Resulting Signal (Test 1).

Implemented Filter

Two different filters were configured for this test. A low pass and a high pass filter. The first one consists of a 4th order low pass composed by two biquad filters. The second one is a 2nd order high pass followed by a 2nd order peak filter. Both filters received the aforementioned signal has their input.

Low Pass Filter

This filter was designed to eliminate the higher frequency component of the input signal while keeping attenuation low for the lower frequency one. As such, the chosen cutoff frequency was 20kHz. Since the 100kHz sub-signal is very dominant, a 4th order filter was required, and consequently two biquad filters were used. Figure 4-17 provides a block diagram of the implemented filter.

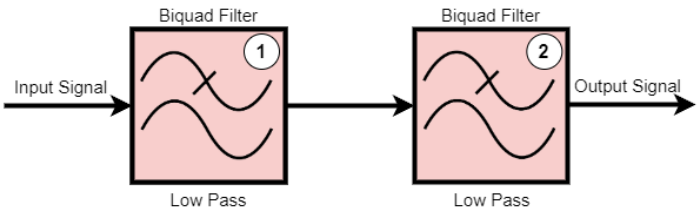


Figure 4-17 – LPF block diagram.

To get a correct Butterworth response out of 2 cascading filters, their Q values were adjusted to 0.5412 and 1.3066 respectively. The resulting transfer function for the biquad first filter was:

① $$H(z) = \frac{0.0035400390625 + 0.007080078125 * z^{-1} + 0.0035400390625 * z^{-2}}{1 - 1.7783203125 * z^{-1} + 0.79241943359375 * z^{-2}} \quad (4.2)$$

While the second filter was:

② $$H(z) = \frac{0.0037841796875 + 0.00750732421875 * z^{-1} + 0.0037841796875 * z^{-2}}{1 - 1.8934326171875 * z^{-1} + 0.908447265625 * z^{-2}} \quad (4.3)$$

The corresponding frequency responses can be observed in Figure 4-18 and Figure 4-19.

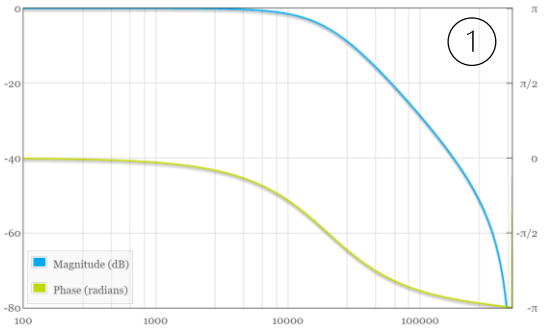


Figure 4-18 – LPF ① frequency response.

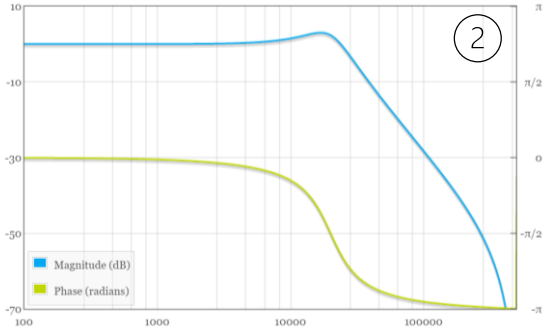


Figure 4-19 – LPF ② frequency response.

High Pass Filter

This filter consists of a 2nd order high pass filter with a cutoff frequency of 100kHz. Since this would mean an attenuation of 3dB to the higher frequency component, a peak filter was implemented to offset this. As expected, this filter has a 3dB gain at 100kHz. Similarly to before, Figure 4-20 shows the resulting block diagram.

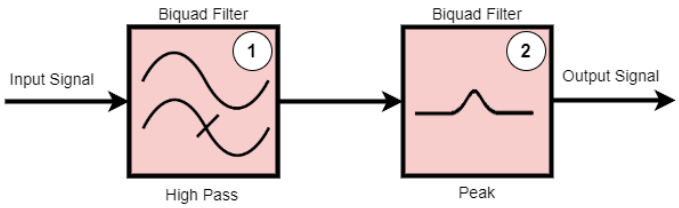


Figure 4-20 – HPF block diagram.

Both the high pass and the peak filters feature a Q value of 0.7071, giving them a Butterworth response. The first biquad filter' transfer function was:

① $$H(z) = \frac{0.638916015625 - 1.27789306640625 * z^{-1} + 0.638916015625 * z^{-2}}{1 - 1.14300537109375 * z^{-1} + 0.41278076171875 * z^{-2}} \quad (4.4)$$

Likewise, the second was:

② $$H(z) = \frac{1.09368896484375 - 1.25048828125 * z^{-1} + 0.4520263671875 * z^{-2}}{1 - 1.25048828125 * z^{-1} + 0.54571533203125 * z^{-2}} \quad (4.5)$$

Figure 4-21 and Figure 4-22 present each filter's frequency response.

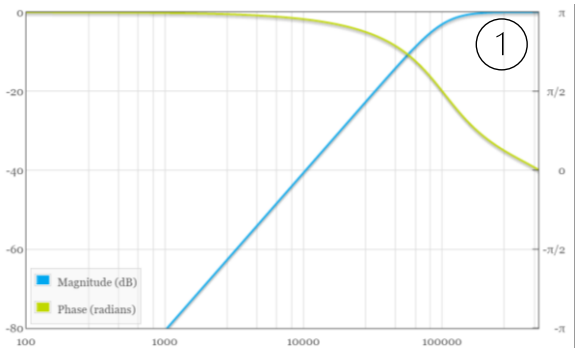


Figure 4-21 – HPF ① frequency response.

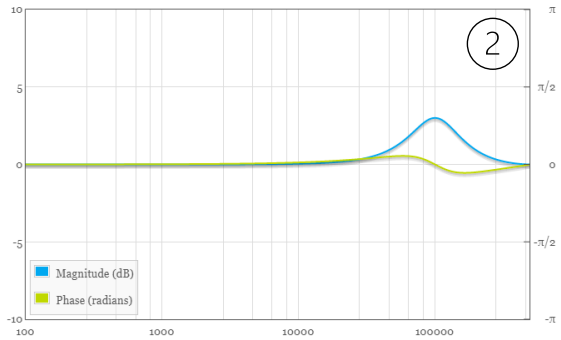


Figure 4-22 – HPF ② frequency response.

Simulated Filter

The expected outputs of both filters were calculated and plotted together with the input signal in Figure 4-23 and Figure 4-24.

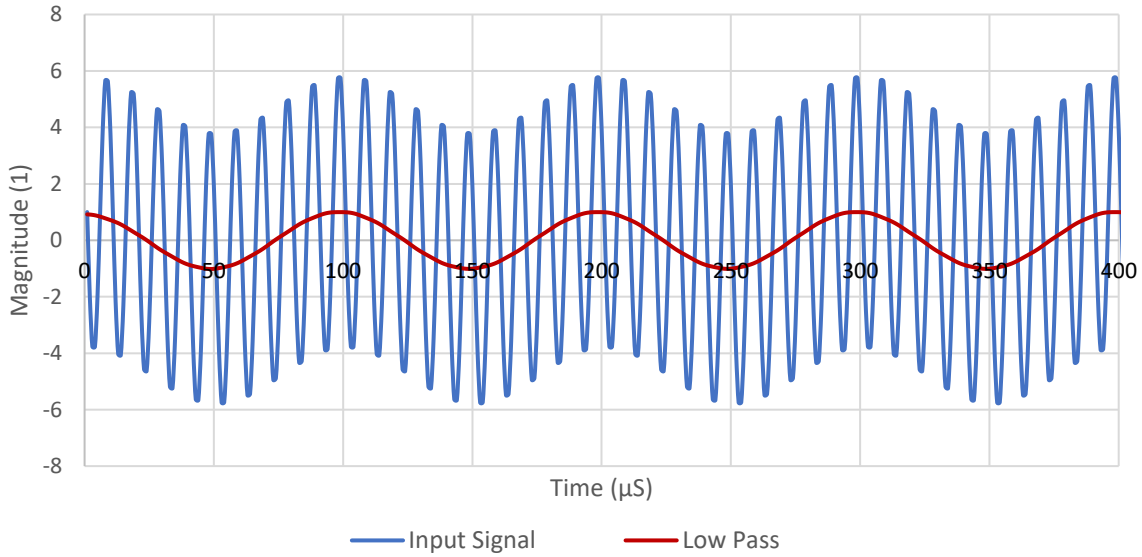


Figure 4-23 – Simulated LPF Results.

From observing Figure 4-23 it's clear that the output signal is the same as the one in the previous Figure 4-14, like intended.

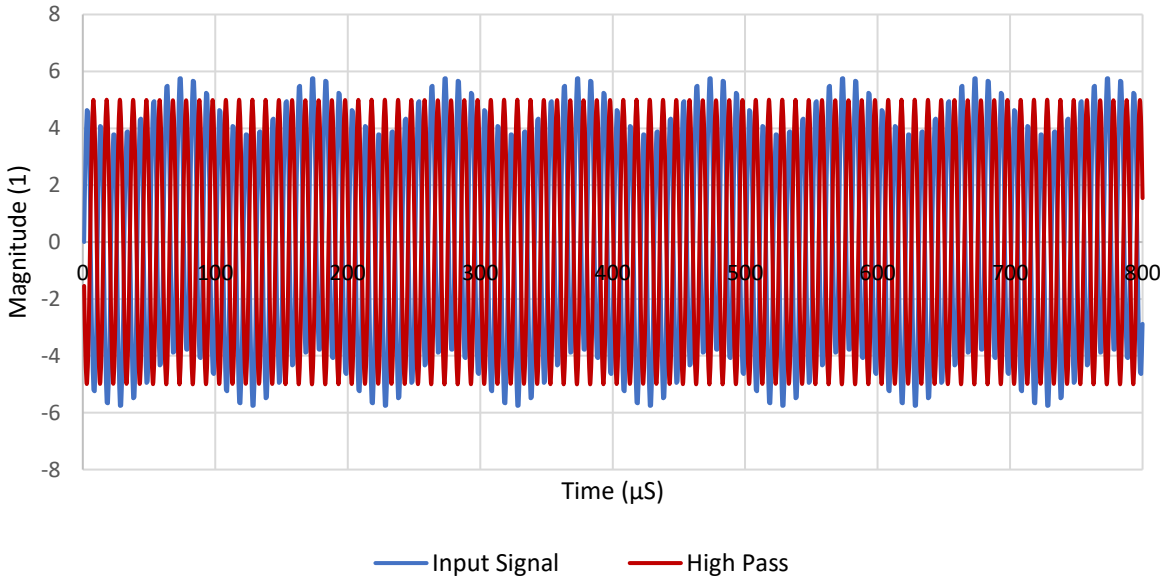


Figure 4-24 – Simulated HPF Results.

Similarly, the output signal from the simulated HPF is the same as Figure 4-15.

Tested Filter Output

Following the previous format, the results obtained from the FPGA implementation are presented in this section, with Figure 4-25 representing the LPF results, and Figure 4-26 the HPF.

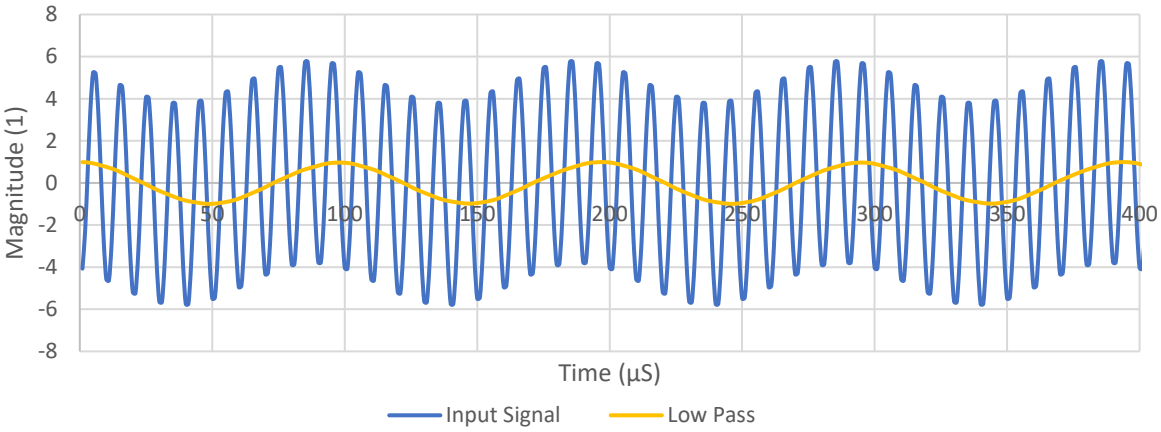


Figure 4-25 – Implemented LPF Results.

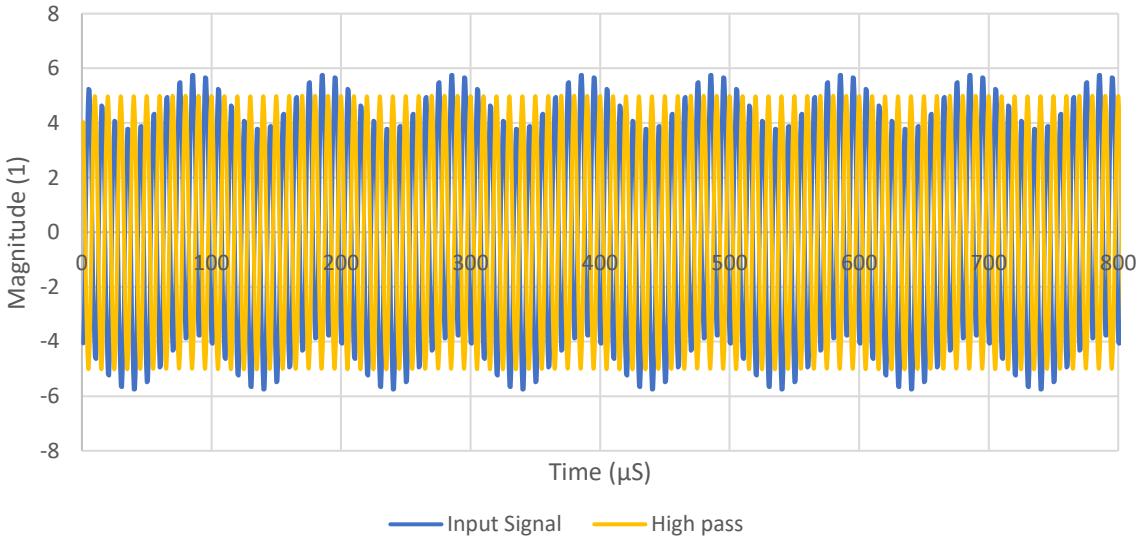


Figure 4-26 – Implemented HPF Results.

It can be observed that both filters' outputs correspond to their simulated counterparts, meaning the behavior was as expected.

Test 2

Setting

This test represents an instance where a signal was sampled with a lot of random noise. To keep matters simple, the signal used is the same as Signal 1 from the previous test. To create the noise signal, random values were assigned from -2 to 2 to each sample, giving it double the maximum magnitude of the sine signal. Since those values of noise are relatively high, a higher order filter is needed. Due to the random nature of noise, both undesired high frequencies and low frequency components will be added to the base signal, meaning a band pass filter is required. The sampling frequency was, again, 1MHz. All signals are documented in the figures below (Figure 4-27, Figure 4-28, and Figure 4-29) .

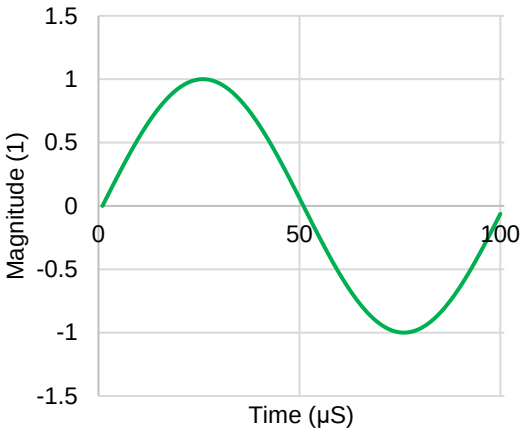


Figure 4-27 – Base Signal (10kHz).

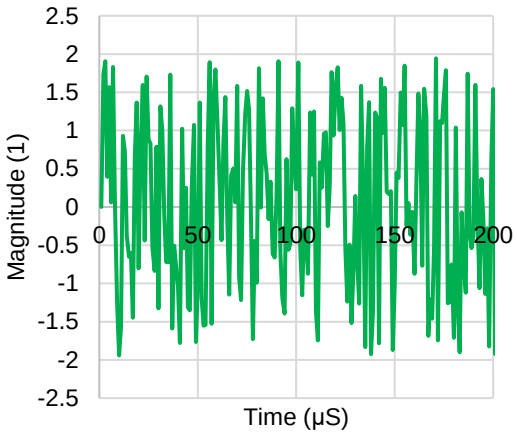


Figure 4-28 – Noise Signal.

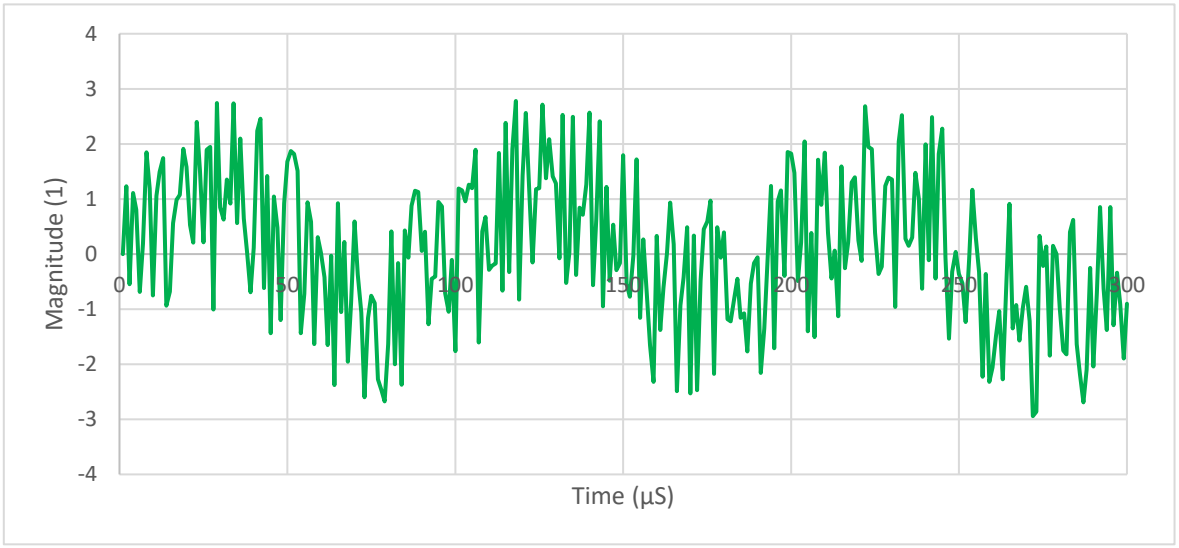


Figure 4-29 – Resulting Signal (Test 2).

Implemented Filter

Unlike the previous test, only one filter was configured this time, a band pass filter. This filter is composed of a 4th order low pass followed by another 4th order high pass, for a total of four biquad modules (Figure 4-30) .

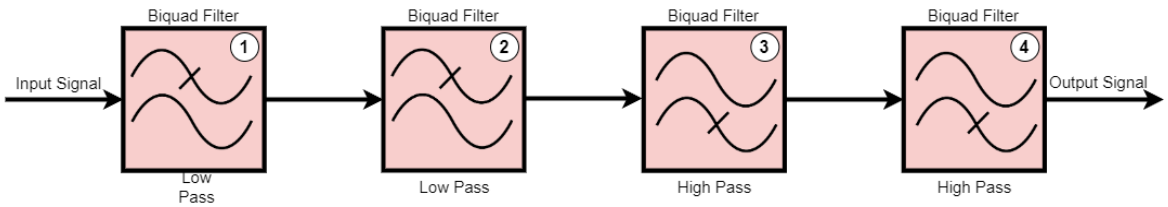


Figure 4-30 – BPF block diagram.

Both the low pass and the high pass filters have their cutoff frequencies set at 10kHz. Like previous 4th order implementations, the Q values were adjusted to maintain a Butterworth response. Below are the transfer functions implemented in each biquad filter:

① $$H(z) = \frac{0.00091552734375 + 0.00189208984375 * z^{-1} + 0.00091552734375 * z^{-2}}{1 - 1.8865966796875 * z^{-1} + 0.89031982421875 * z^{-2}} \quad (4.6)$$

② $$H(z) = \frac{0.0009765625 + 0.001953125 * z^{-1} + 0.0009765625 * z^{-2}}{1 - 1.94921875 * z^{-1} + 0.95306396484375 * z^{-2}} \quad (4.7)$$

③ $$H(z) = \frac{0.9442138671875 - 1.88848876953125 * z^{-1} + 0.9442138671875 * z^{-2}}{1 - 1.8865966796875 * z^{-1} + 0.89031982421875 * z^{-2}} \quad (4.8)$$

④ $$H(z) = \frac{0.9755859375 + 0.001953125 * z^{-1} + 0.9755859375 * z^{-2}}{1 - 1.94921875 * z^{-1} + 0.95306396484375 * z^{-2}} \quad (4.9)$$

For easier comprehension, the frequency responses of all filters are displayed in Figure 4-31 through Figure 4-34.

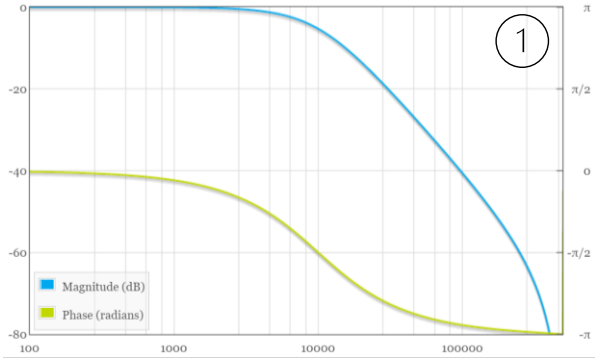


Figure 4-31 – Biquad ① frequency response (test2).

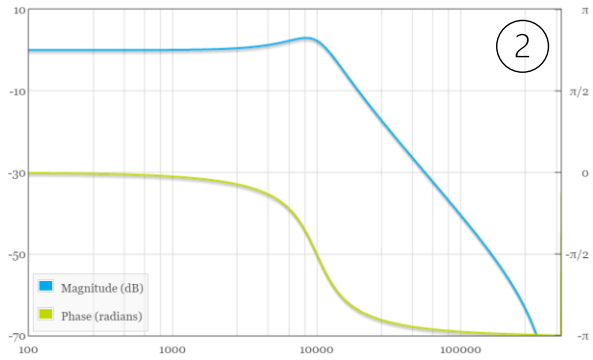


Figure 4-32 – Biquad ② frequency response (test2).

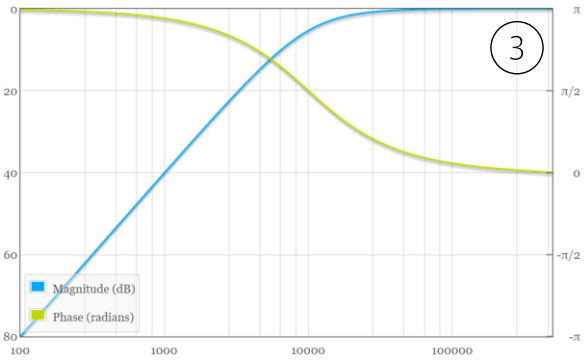


Figure 4-33 – Biquad ③ frequency response (test2).

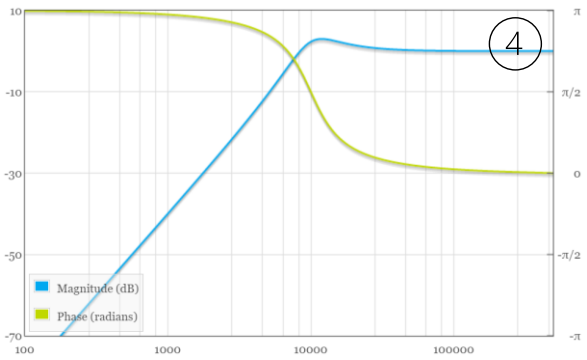


Figure 4-34 – Biquad ④ frequency response (test2).

Simulated Filter

It's expected that the base signal gets attenuated by 6dB due to the effect of both filters (3dB for the low pass, and another 3dB for the high pass). It's also noticeable that it suffered some phase distortion (Figure 4-35) . This is due to the non-linear phase properties of IIR filters like the biquad. There are some ways around this, like using all-pass filters to linearize the phase, or filtering the signal again in reverse. However, such techniques go beyond the scope of this work and therefore weren't implemented.

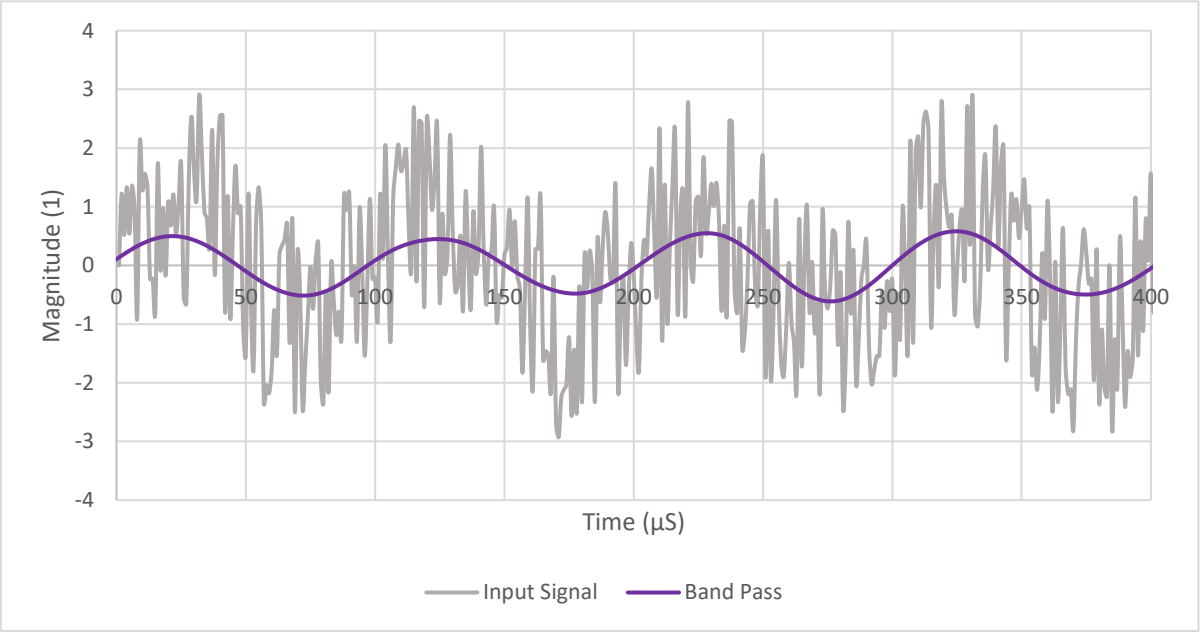


Figure 4-35 – Simulated BPF Results.

Tested Filter Output

The results of the FPGA test were plotted in Figure 4-36.

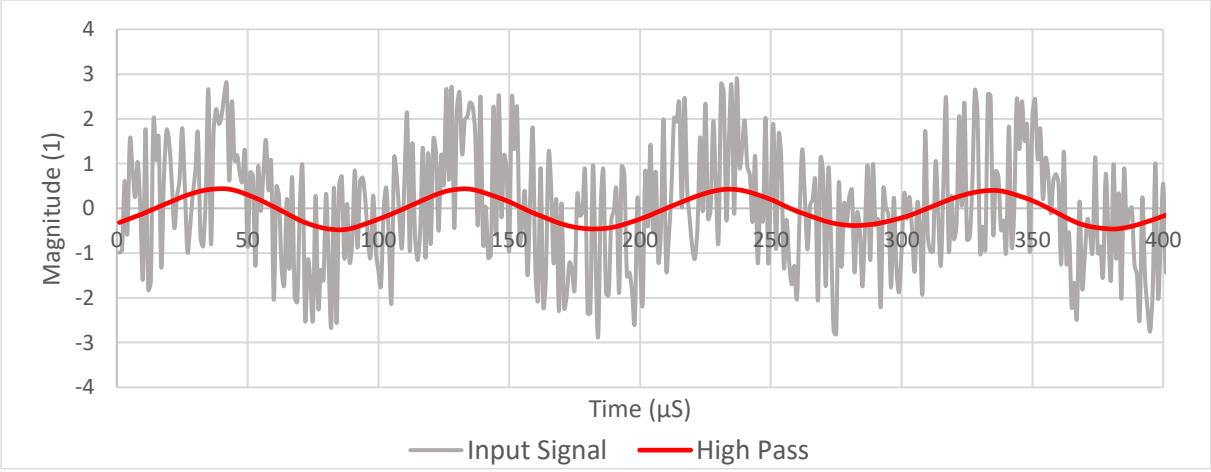


Figure 4-36 – Implemented BPF Results.

It should be noted that the input signal used in this implementation slightly differs from the one in simulation since the noise is random. This in turn means phase distortion will affect these signals a little differently. Nonetheless, albeit with some distortion, the filter was able to salvage the base signal from the noise, like expected.

4.4 Physical Design with Openlane

With the filter properly tested, it was time to begin the physical design using the Openlane flow. This section also serves as a tutorial to SoC integration with Openlane, using the biquad filter and Caravel as an example.

All the steps in the physical design process that are covered in this chapter can be observed in Figure 4-37.

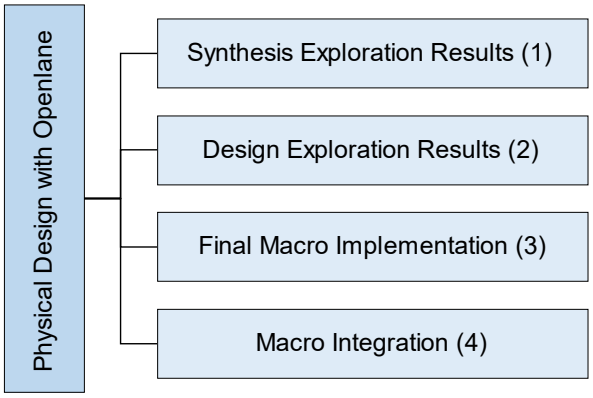


Figure 4-37 – Physical Design steps with Openlane.

The macro hardening process will follow the same steps as those mentioned in the tutorial of the previous chapter 3, although it will focus on the results rather than the process of obtaining them. On the other hand, the macro integration procedure will be covered in more detail here.

4.4.1 Synthesis Exploration Results

The results of the synthesis exploration of the biquad filter are in Figure 4-38.

Best Area	Best Gate Count	Best Delay
60188.98	6667.0	5444.58
AREA2	AREA0	AREA3

Strategy	Gate Count	Area (um^2)	Delay (ps)	Gates Ratio	Area Ratio	Delay Ratio
DELAY0	11597.0	104052.3	7736.07	1.739	1.728	1.42
DELAY1	11280.0	98753.46	7195.14	1.691	1.64	1.321
DELAY2	11394.0	99910.82	7363.84	1.709	1.659	1.352
DELAY3	11803.0	105598.77	7723.69	1.77	1.754	1.418
DELAY4	9152.0	82760.62	9126.93	1.372	1.375	1.676
AREA0	6667.0	60653.17	9505.04	1.0	1.007	1.745
AREA1	6667.0	60678.2	9880.92	1.0	1.008	1.814
AREA2	6670.0	60188.98	9862.93	1.0	1.0	1.811
AREA3	10635.0	74820.51	5444.58	1.595	1.243	1.0

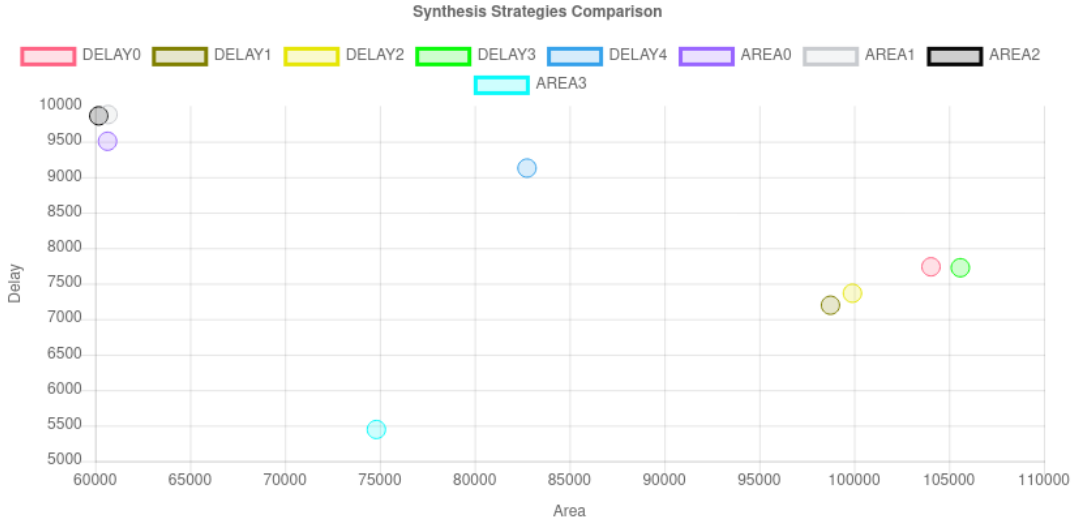


Figure 4-38 – Synthesis exploration dashboard of the biquad design.

From analyzing the dashboard, it becomes clear that the overall best strategy is “AREA3”, since it is the one with the lowest area + delay combination. For this reason, it was the one chosen for this design.

4.4.2 Design Exploration Results

The objective of this phase was to see how far the design’s area could be shrunken while successfully completing the flow with no violations. To achieve this, a regression with the following variable values was performed:

```
FP_CORE_UTIL=(40,43,45,47,50)

PL_TARGET_DENSITY=(0.4,0.45,0.5,0.55)

FP_ASPECT_RATIO=(0.5,0.8,1,1.25,2)
```

It is not expected that this regression yields the optimal combination of parameters, but instead give a clue on where to explore further.

The extracted CSV data was then imported into a sheet-based application and properly analyzed, like observable in Figure 4-39.

design_name	config	flow_status	total_runtime	routed_runtime	(Cell/mm^2)/Core_Util	DIEAREA_mm^2	CellPer_mm^2	OpenDP_Util	Peak_Memory_Usage_MB	cell_count	tritonRoute_violations	Short_viol
bqmain	config_regression_0	flow failed	0h0m13s0ms	-1	-2.5	0.21883789765	-1	41.15	-1	-1	-1	
bqmain	config_regression_1	flow failed	0h0m13s0ms	-1	-2.5	0.2168740192500000	-1	41.11	-1	-1	-1	
bqmain	config_regression_3	flow failed	0h0m13s0ms	-1	-2.5	0.2157985352499999	-1	41.13	-1	-1	-1	
bqmain	config_regression_4	flow failed	0h0m14s0ms	-1	-2.5	0.2154370312500000	-1	41.17	-1	-1	-1	
bqmain	config_regression_2	flow failed	0h0m14s0ms	-1	-2.5	0.216247423425	-1	41.23	-1	-1	-1	
bqmain	config_regression_7	flow completed	0h8m50s0ms	0h5m33s0ms	-2.5	0.216247423425	-1	41.23	1635.76	-1	0	
bqmain	config_regression_6	flow completed	0h8m50s0ms	0h5m30s0ms	-2.5	0.2168740192500000	-1	41.11	1604.31	-1	0	
bqmain	config_regression_5	flow completed	0h8m55s0ms	0h5m26s0ms	-2.5	0.21883789765	-1	41.15	1697.65	-1	0	
bqmain	config_regression_8	flow completed	0h8m48s0ms	0h5m51s0ms	-2.5	0.2157985352499999	-1	41.13	1536.52	-1	0	
bqmain	config_regression_9	flow completed	0h8m57s0ms	0h6m6s0ms	-2.5	0.2154370312500000	-1	41.17	1567.57	-1	0	
bqmain	config_regression_11	flow completed	0h9m4s0ms	0h6m29s0ms	-2.5	0.2168740192500000	-1	41.11	1506.66	-1	0	
bqmain	config_regression_12	flow completed	0h9m10s0ms	0h6m36s0ms	-2.5	0.216247423425	-1	41.23	1575.84	-1	0	
bqmain	config_regression_10	flow completed	0h9m20s0ms	0h6m40s0ms	-2.5	0.21883789765	-1	41.15	1734.95	-1	0	
bqmain	config_regression_20	flow failed	0h0m14s0ms	-1	-2.3255813953488373	0.2041820200000000	-1	44.19	-1	-1	-1	
bqmain	config_regression_21	flow failed	0h0m13s0ms	-1	-2.3255813953488373	0.202286073825	-1	44.16	-1	-1	-1	
bqmain	config_regression_22	flow failed	0h0m29s0ms	-1	-2.3255813953488373	0.2016800448	-1	44.12	-1	-1	-1	
bqmain	config_regression_23	flow failed	0h0m45s0ms	-1	-2.3255813953488373	0.201248806625	-1	44.32	-1	-1	-1	
bqmain	config_regression_24	flow failed	0h1m2s0ms	-1	-2.3255813953488373	0.2009019144000000	-1	44.37	-1	-1	-1	
bqmain	config_regression_13	flow completed	0h9m25s0ms	0h5m20s0ms	-2.5	0.2157985352499999	-1	41.13	1678.94	-1	0	
bqmain	config_regression_14	flow completed	0h9m48s0ms	0h6m19s0ms	-2.5	0.2154370312500000	-1	41.17	1754.08	-1	0	
bqmain	config_regression_15	flow completed	0h9m52s0ms	0h6m25s0ms	-2.5	0.21883789765	-1	41.15	1743.9	-1	0	
bqmain	config_regression_16	flow completed	0h9m53s0ms	0h6m34s0ms	-2.5	0.2168740192500000	-1	41.11	1592.57	-1	0	
bqmain	config_regression_17	flow completed	0h9m51s0ms	0h6m37s0ms	-2.5	0.216247423425	-1	41.23	1705.0	-1	0	
bqmain	config_regression_18	flow completed	0h9m45s0ms	0h6m34s0ms	-2.5	0.2157985352499999	-1	41.13	1608.06	-1	0	
bqmain	config_regression_19	flow completed	0h9m47s0ms	0h6m34s0ms	-2.5	0.2154370312500000	-1	41.17	1724.03	-1	0	
bqmain	config_regression_25	flow completed	0h8m38s0ms	0h5m15s0ms	-2.3255813953488373	0.2041820200000000	-1	44.19	1777.49	-1	0	
bqmain	config_regression_26	flow completed	0h8m23s0ms	0h3m16s0ms	-2.3255813953488373	0.202286073825	-1	44.16	1745.11	-1	0	
bqmain	config_regression_27	flow completed	0h8m27s0ms	0h4m49s0ms	-2.3255813953488373	0.2016800448	-1	44.12	1784.1	-1	0	
bqmain	config_regression_28	flow completed	0h8m20s0ms	0h4m53s0ms	-2.3255813953488373	0.201248806625	-1	44.32	1722.99	-1	0	
bqmain	config_regression_29	flow completed	0h8m22s0ms	0h4m58s0ms	-2.3255813953488373	0.2009019144000000	-1	44.37	1673.15	-1	0	

Figure 4-39 – Raw CSV design exploration data.

The raw data, although already very useful, is complicated and messy to analyze. Therefore, for the sake of this dissertation, Figure 4-40 thought Figure 4-44 were created as a way to offer an easier way to study it. For these, red cells mean the flow failed while green cells mean it succeeded. The values are the resulting die area in mm^2.

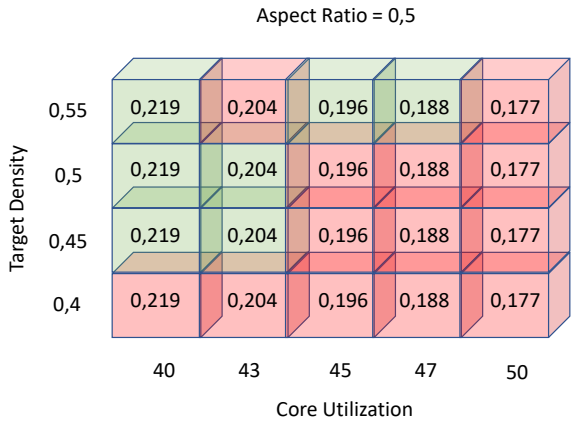


Figure 4-40 – Die area according to TD and CU for AP = 0,5.

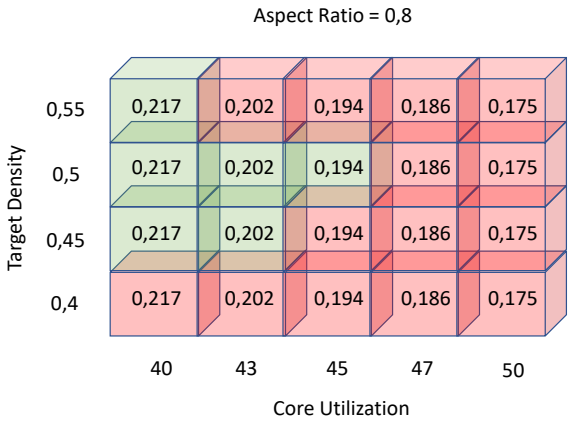


Figure 4-41 – Die area according to TD and CU for AP = 0,8.

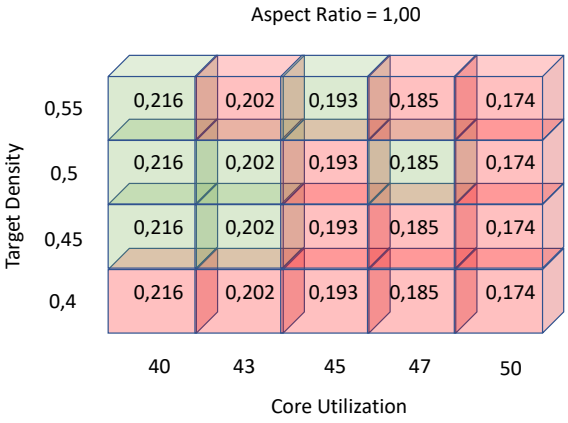


Figure 4-42 – Die area according to TD and CU for AP = 1,0.

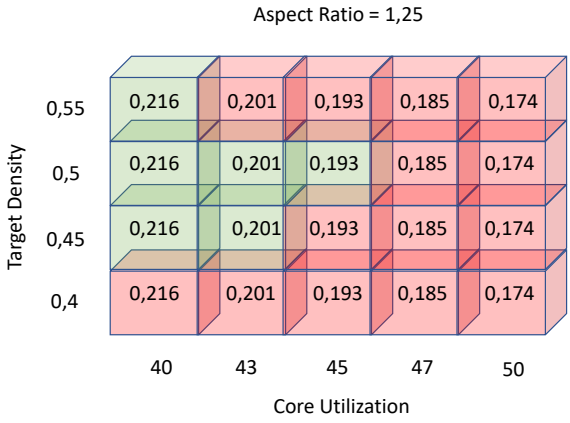


Figure 4-43 – Die area according to TD and CU for AP = 1,25.

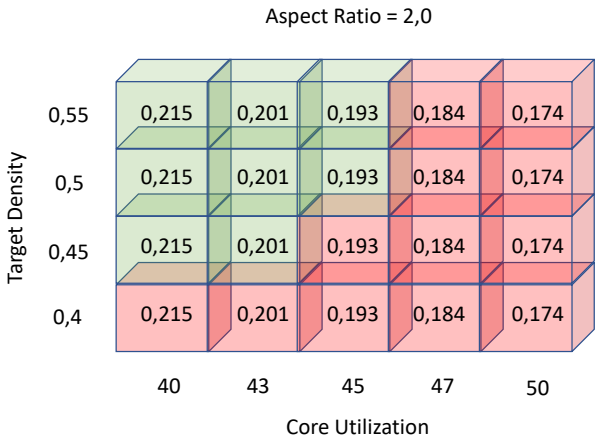


Figure 4-44 – Die area according to TD and CU for AP = 2,0.

The area values in each column of the tables are always the same because they share the same core utilization percentage. However, they differ from table to table. For example, an implementation with Aspect Ratio (AP) = 0,5 and Core Utilization (CU) = 40 uses around 0,219 mm^2, while one with AP = 0,8 uses only 0.217 mm^2. This is because the synthesis tool could not get the exact ratios requested, so some implementations end up using more area than others.

An interesting takeaway is that the tools could not harden a design with Target density (TD) = 0,4 in any condition. This is likely due to the CU being so high, therefore making the die area low, that placing the cells that far apart is impossible.

Only two instances were able to successfully harden with CU = 47, of which the one with AP = 1,0 had the lowest area usage. As such, the combination that reduces area the most is

```
FP_CORE_UTIL=47
PL_TARGET_DENSITY=0.5
FP_ASPECT_RATIO=1.0
```

After further exploration, it was concluded that the previous values provide the best results for the lowest area usage.

4.4.3 Final Macro Implementation

The definitive config.tcl used in the macro hardening process is presented below.

```
set ::env(PDK) $::env(PDK)
set ::env(STD_CELL_LIBRARY) "sky130_fd_sc_hd"

set script_dir [file dirname [file normalize [info script]]]

set ::env(DESIGN_NAME) bqmain

set ::env(VERILOG_FILES) "\
        $::env(DESIGN_DIR)/src/bqmain.v \
        $::env(DESIGN_DIR)/src/biquad.v \
        $::env(DESIGN_DIR)/src/coefio.v \
        $::env(DESIGN_DIR)/src/multa.v \
        $::env(DESIGN_DIR)/src/multb.v"

set ::env(DESIGN_IS_CORE) 0

set ::env(CLOCK_PORT) "wb_clk_i"
set ::env(CLOCK_PORT) "bq_clk_i"
set ::env(CLOCK_PERIOD) "10"

set ::env(SYNTH_STRATEGY) "AREA 3"
set ::env(SYNTH_MAX_FANOUT) 7

set ::env(FP_CORE_UTIL) 47
set ::env(PL_TARGET_DENSITY) 0.5
set ::env(FP_ASPECT_RATIO) 1

set ::env(ROUTING_CORES) 12
set ::env(FP_PIN_ORDER_CFG) $::env(DESIGN_DIR)/pin_order.cfg

set ::env(RT_MAX_LAYER) {met4}

set ::env(VDD_NETS) [list {vccd1}]
set ::env(GND_NETS) [list {vssd1}]

set ::env(DIODE_INSERTION_STRATEGY) 4

set ::env(RUN_CVC) 1
```

It's worth noting that the pins of the macro were not randomly assigned, but rather set beforehand to help with the SoC integration. Since Caravel's wishbone and logic analyzer ports are on the south side of the chip, all pins of the macro that would connect to them are also on the south. On the other hand, pins that would be connected to the I/O ports of the SoC are on the east side of the macro. All other pins

were connected to the north. By previously organizing pin placement like so, it reduces congestions after integration. Figure 4-45 gives a view of the final GDS file for the biquad filter while Figure 4-46 provides a placement density analysis.

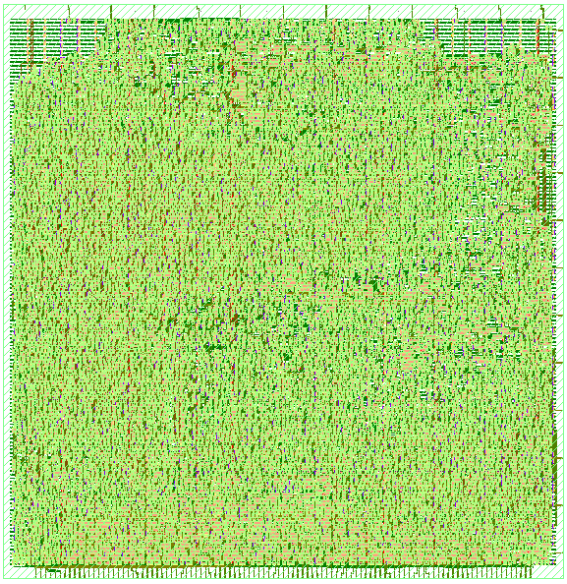


Figure 4-45 – Biquad filter final GDS.

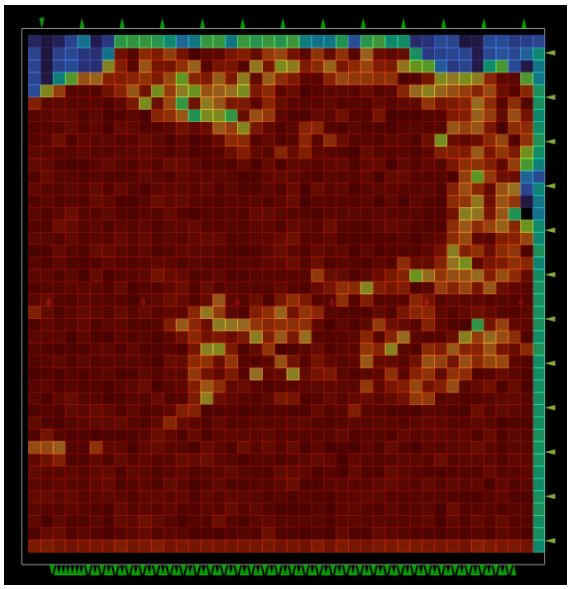


Figure 4-46 – Biquad filter placement density heatmap.

Like expected from the design exploration, the die area is almost all used making this a very efficient implementation.

4.4.4 Macro Integration

To help with the macro integration on Caravel, Efabless provides users with a repository containing a demo project. There are tags for every MPW version, each with their corresponding version of Openlane and sky130 PDK installed. This is meant to save time for users that only want to participate in the open shuttles [34].

It is also possible to harden macros the same way as the normal installation of Openlane, although the command is now

```
make <design>
```

Despite still being possible with some tinkering, synthesis and design exploration are harder to do since they are not an option in the make files. Since the average user is not proficient with them or scripting languages like Python and TCL, it is advisable to use the normal version to perform those explorations.

There are three important files that must be modified in order to integrate a macro into Caravel:

- Top Verilog (user_project_wrapper): can be found inside the "verilog/rtl" directory. This is where users will instantiate their designs and interface them with Caravel at a logic level.

- Configuration (config.tcl): it is located in “openlane/user_project_wrapper”. This defines the flow variables and references the Verilog, GDS and LEF files of the macros.
- Macro placement (macro.cfg): situated in the same folder as config.tcl, this file defines where the macros will be placed in the Caravel user space (floorplanning).

After configuring each file correctly, Openlane hardened the design again which resulted in the GDS file shown below.

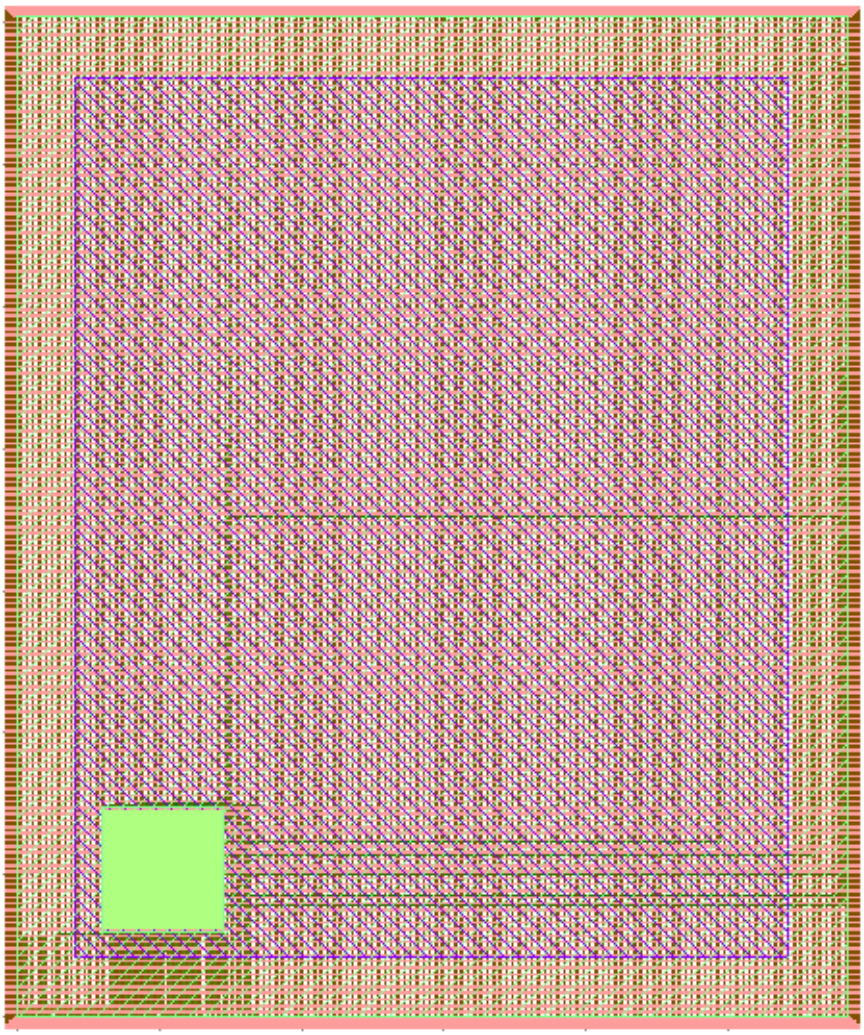


Figure 4-47 – View of the user project GDS.

In Figure 4-47 we can clearly see that the macro was successfully integrated into the user area and connected to the I/O ports. This design will then be combined into the Caravel SoC by Efabless before fabrication.

The last step is to pass the pre-check, both locally and on the Efabless website. To do so, users need to run

```
make run-precheck
```

If all checks are passed, the project is eligible for submission in the open MPW program. Figure 4-48 shows the submitted project in the Efabless page along with other MPW-7 projects [35].

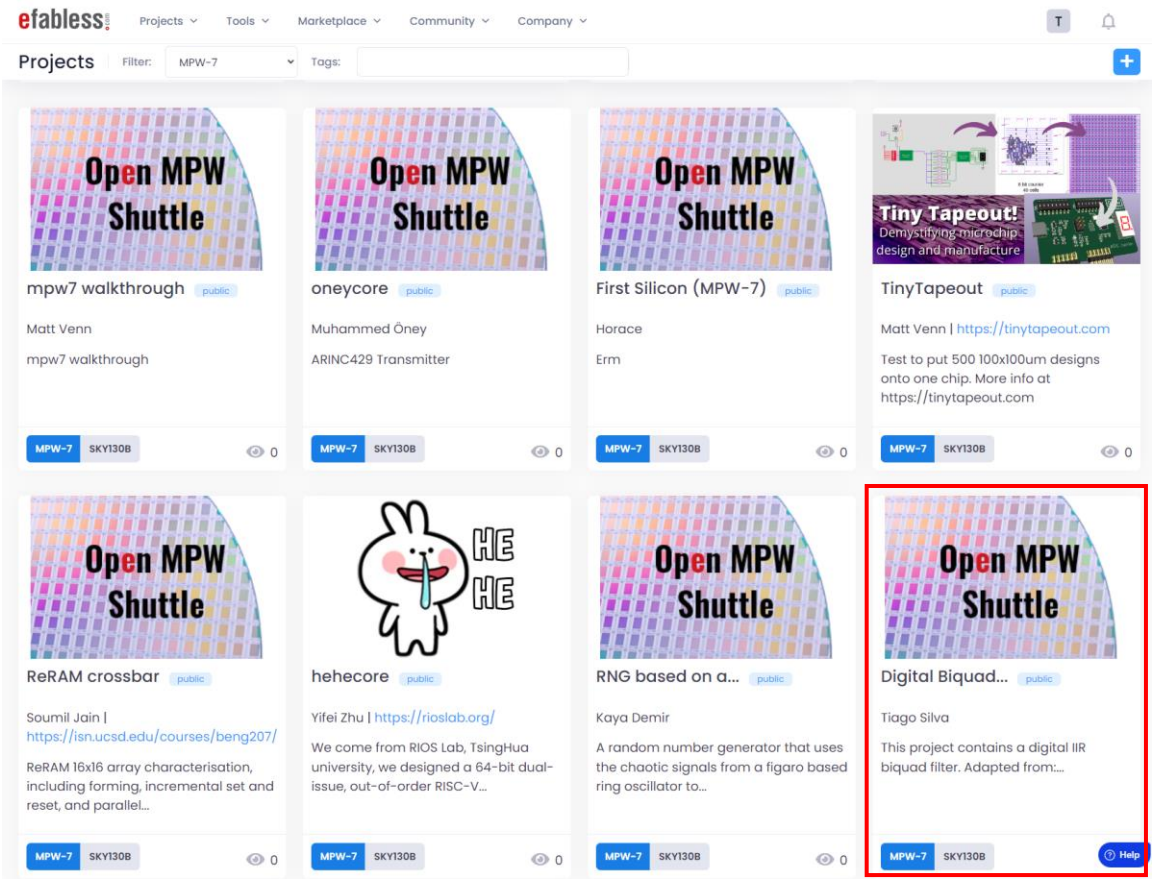


Figure 4-48 – Submitted project.

5 CONCLUSION AND FUTURE WORK

Over the course of this dissertation various open-source tools were presented and studied. Then, in the last chapter, a design flow was proposed based on such studies and a practical example was given. This section presents a conclusion to this work, highlighting the biggest challenges faced, and future work that could be done in continuation to this ambitious project.

5.1 Conclusion and Challenges Faced

The open-source movement has clearly reached the hardware industry, specifically the chip design one. With the recent SKY130 open PDK and Openlane tools, together with RISC-V they provide the possibility of having the first truly open design flow for ASIC. However, the freshness of these tools is still felt when compared proprietary ones. The rapid pace at which they are evolving brings instability to the framework. With new bugs and problems being both fixed and added every day, even the task of setting up the environment can be arduous and painful. Nevertheless, the improvement over time is undeniable and so is the growing community. RISC-V on the other hand, although still not as prevalent as its main competitor ARM, shows much more maturity and has already been widely adopted proving that it is in the fact the ISA of the future.

The proposed design flow demonstrates an open alternative to proprietary tools and lowers the barrier on entry to chip design. It is especially useful in the fields of education since it allows all students and researchers to get hands on experience with a framework that they might otherwise not have access to. There is a caveat to this, however, and that is the cost of the FPGA. This work also showed the importance of hardware verification before Tapeout, making it almost a mandatory step in the process.

To ascertain the feasibility of the proposed flow, the first component of a SoC was developed, tested, hardened, and integrated into Caravel. Through some optimization processes, the final biquad filter's area was reduced to 0,185 mm^2 out of the available 10,2784 mm^2, leaving roughly 98.2% of the user area free for other modules to be added in the future. Aside from these results, a tutorial on the Openlane tools was provided.

5.2 Future Work

With the groundwork already done, and a design flow defined, the rest of the digital blocks missing in the SoC proposed in 4.2 can be quickly hardened and integrated. The AFE is also missing, but developing it would require the use of open analog EDA tools such as Xschem and Ngspice, which were not covered in this project. A further study into these alongside a proposed analog and mixed signal design flow are needed and highly significant.

Another valuable contribution would be porting the Swervolfx system to other FPGAs, and most importantly adapt its Verilog so that it is compatible with ASIC. This would enable the use of the same system throughout logic and physical design flows. Alternatively, the Caravel SoC could be converted to work on a FPGA.

5.3 Final Words

Having worked in the open-source chip design space for a year, it became clear that it is a movement that is here to stay. Despite that, and even if the existing tools are capable of producing impressive results, they are still not mature enough to be realistically used in educational settings, let alone in commercial ones. However, that is to be expected. After all, rapid growth is always accompanied by “growing pains”. When these subdue, and the tools start getting more stable, Openlane and Skywater’s open PDKs will certainly become a staple in chip design education.

BIBLIOGRAPHY

[1] M. P. FORRER, “Survey of Circuitry for Wristwatches,” vol. 60, no. 9, 1972.

[2] “Company - Infineon Technologies.” https://www.infineon.com/cms/en/about-infineon/company/ (accessed Sep. 26, 2022).

[3] “PSoC™ 64 - Secured MCU - Infineon Technologies.” https://www.infineon.com/cms/en/product/microcontroller/32-bit-psoc-arm-cortex-microcontroller/psoc-6-32-bit-arm-cortex-m4-mcu/psoc-64/ (accessed Sep. 26, 2022).

[4] “Zynq UltraScale+ RFSoC.” https://www.xilinx.com/products/silicon-devices/soc/rfsoc.html (accessed Sep. 26, 2022).

[5] A. Pandey, “Simulating a pipelined RISC processor,” *Proc. Int. Conf. Inven. Comput. Technol. ICICT 2016*, vol. 2, 2016, doi: 10.1109/INVENTIVE.2016.7824854.

[6] C. H. Séquin and D. A. Patterson, “Design and Implementation of RISC-I,” vol. 38, no. 4. pp. 858–864, 2009.

[7] “RISC-V Instruction Set Manual, Volume I: RISC-V User-Level ISA | Five EmbedDev.” https://five-embeddev.com/riscv-isa-manual/latest/history.html (accessed Feb. 25, 2022).

[8] “RISC-V International.” https://riscv.org/ (accessed Feb. 25, 2022).

[9] E. Blem, J. Menon, V. Thiruvengadam, and K. Sankaralingam, “ISA Wars: Understanding the Relevance of ISA being RISC or CISC to Performance, Power, and Energy on Modern Architectures.” ACM Trans. Comput. Syst. 33, 1, Article 3, p. 34, 2015, doi: 10.1145/2699682.

[10] M. Ling, X. Xu, Y. Gu, and Z. Pan, “Does the ISA Really Matter? A Simulation Based Investigation,” *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM 2019 - Proceedings*. 2019, doi: 10.1109/PACRIM47961.2019.8985059.

[11] C. Price, “MIPS IV Instruction Set,” *Memory*, 1995, [Online]. Available: http://www.weblearn.hs-bremen.de/risse/RST/docs/MIPS/mips-isa.pdf.

[12] D. Weaver, “The SPARC architecture manual V8,” *ReVision*, 1990, [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.2805&rep=rep1&typ

e=pdf.

[13] Ibm, “Power ISA (v2.05),” *Power*, 2007.

[14] ARM, “ARM Architecture Reference Manual,” pp. 1–1138, 2007.

[15] M. J. Cannizzaro, E. W. Gretok, and A. D. George, “RISC-V benchmarking for onboard sensor processing,” *Proceedings - 2021 IEEE Space Computing Conference, SCC 2021*. pp. 46–59, 2021, doi: 10.1109/SCC49971.2021.00013.

[16] C. Imianosky, P. R. O. Valim, C. A. Zeferino, and F. Viel, “Evaluating the CCSDS 123 Compressor Running on RISC-V and ARM Architectures,” *Brazilian Symposium on Computing System Engineering, SBESC*, vol. 2020-Novem. 2020, doi: 10.1109/SBESC51047.2020.9277854.

[17] S. Upgrades, R. L. After, and J. Six, “P650 P UMPS U P P ERFORMANCE BY 50 %,” no. December 2021, 2023.

[18] “HiFive Unmatched - SiFive.” https://www.sifive.com/boards/hifive-unmatched (accessed Feb. 25, 2022).

[19] “SiFive HiFive Unmatched Hands-On, Initial RISC-V Performance Benchmarks - Phoronix.” https://www.phoronix.com/scan.php?page=article&item=hifive-unmatched-benchmarks&num=1 (accessed Feb. 25, 2022).

[20] “SparkFun RED-V RedBoard - SiFive RISC-V FE310 SoC - DEV-15594 - SparkFun Electronics.” https://www.sparkfun.com/products/15594 (accessed Feb. 25, 2022).

[21] “SparkFun RED-V Thing Plus - SiFive RISC-V FE310 SoC - DEV-15799 - SparkFun Electronics.” https://www.sparkfun.com/products/15799 (accessed Feb. 25, 2022).

[22] “LOFIVE-R1 GroupGets LLC | Development Boards, Kits, Programmers | DigiKey.” https://www.digikey.pt/en/products/detail/groupgets-llc/LOFIVE-R1/10186935 (accessed Feb. 25, 2022).

[23] “Sipeed Maixduino Kit for RISC-V AI + IoT - Seeed Studio.” https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html (accessed Feb. 25, 2022).

[24] “Variables information — OpenLANE documentation.” https://openlane-docs.readthedocs.io/en/rtd-develop/configuration/README.html (accessed Sep. 26, 2022).

[25] “SkyWater Foundry Provided Standard Cell Libraries — SkyWater SKY130 PDK 0.0.0-299-g79d35fe documentation.” https://antmicro-skywater-pdk-docs.readthedocs.io/en/test-submodules-in-rtd/contents/libraries/foundry-provided.html (accessed Sep. 30, 2022).

[26] “SkyWater and Google expand open source program to new 90nm technology | Google Open Source Blog.” https://opensource.googleblog.com/2022/07/SkyWater-and-Google-expand-open-source-program-to-new-90nm-technology.html (accessed Sep. 26, 2022).

[27] “efabless.com.” https://efabless.com/open_shuttle_program (accessed Sep. 26, 2022).

[28] “Caravel Management SoC - Litex — Caravel Management SoC documentation.” https://caravel-mgmt-soc-litex.readthedocs.io/en/latest/ (accessed Sep. 26, 2022).

[29] “Chips Alliance - Chips Alliance.” https://chipsalliance.org/ (accessed Sep. 30, 2022).

[30] “Verilator User’s Guide — Verilator 4.225 documentation.” https://verilator.org/guide/latest/index.html (accessed Sep. 30, 2022).

[31] “GTKWave.” https://gtkwave.sourceforge.net/ (accessed Sep. 30, 2022).

[32] A. Roy *et al.*, “A 6.45 µ W Self-Powered SoC with Integrated Energy-Harvesting Power Management and ULP Asymmetric Radios for Portable Biomedical Systems,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 6, pp. 862–874, 2015, doi: 10.1109/TBCAS.2015.2498643.

[33] “Imagination University Programme.” https://university.imgtec.com/ (accessed Sep. 30, 2022).

[34] “GitHub - efabless/caravel_user_project: https://caravel-user-project.readthedocs.io.” https://github.com/efabless/caravel_user_project (accessed Sep. 30, 2022).

[35] “Biquad Project Detail | Efabless.” https://platform.efabless.com/projects/1243 (accessed Sep. 30, 2022).